Escola Politècnica Superior de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

TRABAJO FINAL DE CARRERA

TÍTULO: Diseño y desarrollo de un sistema de diagnóstico para equipos basados en dispositivos programables

AUTOR: José M. Barreiro García

DIRECTOR: Jordi Oriol Giró

SUPERVISOR: Marcos Quílez Figuerola

FECHA: 5 de Septiembre de 2005

Título: Diseño y desarrollo de un sistema de diagnóstico para equipos basados en dispositivos programables.

Autor: José M. Barreiro García

Director: Jordi Oriol Giró

Supervisor: Marcos Quílez Figuerola

Fecha: 5 de septiembre de 2005

Resumen

Este documento describe de manera detallada el análisis, diseño, y desarrollo de un sistema capaz de testear de manera eficiente, placas electrónicas compuestas principalmente por dispositivos programables.

El sistema se ha probado sobre una único modelo de placa, pero se ha diseñado para que sea aplicable a cualquier otro PCB (printed circuit boars), con el único requisito de que algunos de sus dispositivos lleven implementado el estándar *IEEE 1149.1 boundary-scan test.*

En la primera parte de éste documento se realiza un estudio sobre el hardware de uno de los módulos de aplicación del equipo Victoria Combo de Trend Communications S.L. para, a continuación, establecer los objetivos a cumplir por el sistema de diagnóstico, y elegir alguna de las posibles vías de desarrollo.

En la segunda parte se describe el diseño y desarrollo de las aplicaciones que componen el sistema, adjuntándose los diagramas de flujo y códigos fuente.

Por último, se exponen y evaluan los resultados obtenidos tras aplicar el sistema de diagnóstico sobre una placa modificada, simulando así los principales defectos de montaje que se producen durante la fabricación de las placas.

Title: Design and development of a diagnosis system for equipment based on programmable devices.

Author: José M. Barreiro García

Director: Jordi Oriol Giró

Supervisor: Marcos Quílez Figuerola

Date: September, 5th 2005

Overview

This document describes in detail the analysis, design and development of a system capable of testing electronic printed circuit boars (PCBs) in an efficient way. These PCBs are mainly composed by programmable devices.

The system has been tested on one model of PCB. However, it has been designed to be applicable to any other type of PCB with some devices implemented following the IEEE 1149.1 specification. The mentioned standard refers to boundary-scan testing.

In the first part of this document, a study of the hardware of one test module from Victoria Combo, a Trend Communications S.L. tester, is carried out. Afterwards, the objectives that the diagnosis system has to support are established in order to select a right development procedure.

In the second part, the design and development of the applications composed by the system are described. The flow charts and source codes are also included.

At the end of the document, the results after working with the diagnosis system on a modified PCB are exposed and evaluated. In this way, the main assembly faults produced during the manufacture of PCBs are simulated.

ÍNDICE

INTRODUCCIÓN

En los últimos años, la necesidad de conseguir cada vez velocidades de transmisión mayores, ha obligado a los ingenieros a desarrollar los equipos con la tecnología mas puntera del mercado, y a utilizar dispositivos cada vez mas potentes y con mayor capacidad de cálculo.

Esta mayor capacidad de cálculo, se ve reflejada en los nuevos encapsulados para los dispositivos programables, utilizándose cada vez con mas frecuencia el tipo BGA (*Ball Grid Array*).

Este tipo de encapsulado se caracteriza principalmente porque todas sus conexiones se encuentran en la superficie inferior del dispositivo, lo que provoca que en el proceso de montaje, sea imposible verificar la correcta soldadura mediante una inspeccion visual.

Sin embargo, existe la necesidad de comprobar de una manera fiable la correcta conexión de todos sus terminales, ya que en ellos se concentran la mayoria de lineas de datos del PCB (printed circuit boars).
En caso de mal funcionamiento, se debe descartar con suficiente garantia que no se debe a alguna de las piezas BGAs, o identificar de cual de ellas se trata si esta fuera la causa.
Actualmente existen en el mercado herramientas que realizan estas funciones.

No disponer de este tipo de herramientas, deja como únicas soluciones resoldar las piezas, o reemplazarlas por nuevas, ambas con riesgos de destrucción para los dispositivos, coste elevado, y sin garantias de solucionar el problema.

Trend Communications S.L. diseña, fabrica y vende equipos de medida para comunicaciones digitales SDH (*Synchronous Digital Hierarchy***)** y PDH (*Plesiochronous Digital Hierarchy*).
Estudia desde hace tiempo la posibilidad de adquirir una de estas herramientas, pero su elevado coste, a llevado a ésta a retrasar dicha adquisición, y a intentar solventar sus necesidades utilizando otros recursos.

Por lo tanto, se hace evidente la necesidad de desarrollar un sistema capaz de detectar fallos de montaje sin poner en riesgo los dispositivos, y que permitan descartar o actuar sobre algún elemento en concreto, minimizando costes innecesarios.

Se definen por tanto los objetivos de dicho TFC, que consiste en el diseño y desarrollo de un sistema de diagnóstico capaz de:

- Detectar cualquier tipo de error de montaje que se produzca durante el proceso de frabricación.

- Una vez detectado, debe informar al usuario de dicho fallo, y facilitar la información suficiente para que pueda actuar en consecuencia.

- El sistema debe ser flexible y aplicable a cualquier placa de características similares, permitiendo su aplicación (en un futuro) al resto de placas del grupo.

- El sistema debe ser capaz de realizar el test en un tiempo que no afecte demasiado a otros procesos de fabricación o diseño.

- La inversión en dicho sistema debe ser mínimo, utilizando los recursos de que dispone la empresa, aunque se estudiará la viabilidad de posibles adquisiciones que mejoren la eficiencia del sistema.

Tras un análisis del hardware y de las posibilidades que ofrecen los dispositivos programables de la placa Next Generation del equipo Victoria Combo (utilizada para el desarrollo del sistema), se decide utilizar el estándar *IEEE 1149.1 boundary-scan test.*

Este estándar, da acceso a la mayoría de puertos del dispositivo, mediante cuatro lineas de control, y permite la ejecución de una rutina de test, capaz de detectar el mal conexionado de sus lineas de datos.

Se trata por tanto de desarrollar un conjunto de aplicaciones, que utilizando un fichero con la información sobre el conexionado de la placa, ejecute una serie de acciones que permita la detección de errores montaje.

Se decide dividir el sistema en dos aplicaciones totalmente independientes.

- La primera de ellas, utiliza un fichero de netlist para identificar aquellas nets (*líneas de datos*) que pueden ser testeadas de manera segura (sin riesgos de destrucción para los dispositivos), y la interconexión entre los distintos elementos de la placa.
  A continucación, genera las instrucciones (vectores de test) necesarias para la comprobación de estas, y las almacena en un fichero de texto.

- La segunda aplicación, recoje los vectores de test del fichero de texto y los ejecuta sobre los dispositivos.
  Tras concluir el test, muestra por pantalla y almacena en un fichero de texto los resultados, indicando (en caso de fallo) los dispositivos defectuosos y sus posibles causas.

En los capítulos centrales de este documento, se describen las funciones que realizan cada una de las aplicaciones, y se adjuntan los diagramas de flujo y códigos fuente de las rutinas desarrolladas.

Por ultimo, se han realizado una serie de pruebas de funcionamiento, ejecutando el sistema de diagnóstico sobre una placa modificada, para simular los errores de montaje que se producen durante el proceso de fabricación.

Dichas pruebas consisten en modificar los estados lógicos de algunas nets, y comprobar que el sistema es capaz de detectar el fallo, mostrarlo por pantalla, y facilitar la suficiente información para que el usuario identifique la causa.

Otro de los objetivos de estas pruebas es valorar la eficiencia del sistema, comprobando el porcentaje de placa testeada, así como el tiempo utilizado.

Estos datos servirán para extraer conclusiones sobre el desarrollo del sistema, y plantear posibles mejoras para posteriores versiones.

CAPÍTULO 1. PLATAFORMA VICTORIA COMBO

1.1 Equipo de Test Victoria Combo

1.1.1 Trend Communications

Trend Communications (www.trendcomms.com) es una empresa dedicada al diseño, creación y venta de equipos de medida para diferentes servicios como son banda ancha, voz, datacomm, gestión de red, redes metropolitanas, móviles, y transmisiones ópticas.

Para estas últimas, Trend Communications cuenta con una amplia familia de analizadores diseñados especialmente para la instalación y mantenimiento de redes SDH (*Synchronous Digital Hierarchy*).

1.1.2 Plataforma Victoria Combo

Entre estos equipos cabe destacar el Victoria Combo, un equipo de pruebas portable concebido para analizar y evaluar las redes y los equipos SDH, SONET, PDH y T-Carrier. Es un instrumento completamente modular que permite desde la medida más simple a la más sofisticada, cubriendo todas las necesidades de los ingenieros de test y medida.

Victoria Combo incorpora todas las interfaces de la UIT-T y Bellcore hasta 10 Gbit/s.

SDH	SONET	PDH	T-Carrier
STM64	OC192	E4	DS3
STM16	OC48	E3	DS1
STM4	OC12	E2	nx56/64kbit/s
STM1	OC3	E1	
STM0	STS3-3c	nx64kbit/s	
G832	STS1		

Fig. 1.1 Interfaces de la UIT-T y Bellcore

Entre sus características destacan:

- Hasta 8 módulos con combinaciones de la misma o de varias tecnologías
- El diseño de Victoria Combo es apilable y modular
- Incluye todas las interfaces SDH/SONET hasta 10 Gbit/s
- Conectores ópticos fáciles de limpiar
- Control remoto por internet con un navegador estándar
- Conectividad universal: LAN, inalámbrico, USB, Serie...
- Dispositivos de memoria extraíbles para transferir y almacenar fichero.
- Modo *Sleep* facilita un arranque inmediato ahorrando tiempo y energía
- Baterías de carga rápida
- Batería adicional.

Sus aplicaciones principales son:

- Realiza todas las pruebas imaginables, incluyendo TCM, APS, movimientos de puntero, Alarmas M/N, G.828 y G.829, entre otras
- Pruebas BER en DCC y resto canales de cabecera
- Estadísticas de bloques FEC G.707 para SDH
- Pruebas automáticas Pasa/Falla
- Medida de potencia óptica sin necesidad de instrumentos adicionales
- Pruebas de estrés con desviación de frecuencia
- Trazados detallados con funciones búsqueda y cuantificación
- Captura de tramas programable para trazar protocolos
- Salida de reloj para sincronización
- Medidas en servicio y fuera de servicio.

Fig. 1.2 Plataforma Victoria Combo

1.2 Módulo Next Generation 2,5 Gbits/s

1.2.1 Módulo NG 2.5G

Uno de los módulos disponibles para Victoria Combo es el **NG 2.5G.**

Permite el análisis, monitorización y mantenimiento de las redes SDH/SONET de próxima generación desde 52 Mbit/s hasta 2.5 Gbit/s, y cuyas características principales son las siguientes:

Próxima generación SDH/SONET

- Análisis / emulación de encapsulaciones GFP
- Análisis / emulación de concatenación VCAT
- Análisis / emulación de protocolos LCAS
- Análisis / emulación de EoS
- LO-VCAT (hasta 64 contenedores)
- HO-VCAT (hasta 256 contenedores)
- Control de retardos diferenciales hasta 256 ms
- Soporte completo de eventos de SDH clásico y de próxima generación
- Pruebas BER en las capas física, GFP y Ethernet / IP
- Soporte de Ethernet MAC
- Análisis según RFC-2544
- Generación de tramas Ethernet
- BER Ethernet
- Análisis de carga útil de múltiples capas (LLC, VLAN, etc.)
- IP ping

Aplicaciones principales del módulo **NG 2.5G**

- Migración hacia las redes SDH/SONET de próxima generación
- Instalación de nodos de multiservicio (MSPP)
- Pruebas completas para las redes clásicas SDH/SONET
- Resolución de averías utilizando el trazado con búsqueda / cuantificación
- Captura de tramas programable para analizar los protocolos de transmisión
- Ideal para trabajo de campo

1.2.2 El Hardware del NG2.5G

El NG2.5G se compone principalmente de dos PCB, denominados ADPOW y MAIN.

La primera de ellas (ADPOW) se encarga de suministrar la energía necesaria a la placa MAIN, y por lo tanto, se compone principalmente de fuentes de alimentación conmutadas y conectores.
Su complejidad y probabilidad de fallo son bajos, debido al mínimo número de componentes.

Por el contrario, la placa MAIN es mucho más compleja.
En esta placa se concentra todo el procesado, control, conectividad, y transductores óptico-eléctricos.

Esta formada por unos 2000 componentes y 1800 líneas de transmisión de datos (en adelante "***nets***") distribuidas en 10 capas.

1.3 Necesidades de Test

1.3.1 Análisis del Hardware

Como hemos comentado anteriormente, el número de componentes es importante, y por tratarse de un módulo tecnológicamente avanzado, la mayoría de ellos son de un **coste elevado**.

Para el procesado de señales, y la generación de tramas, se utilizan FPGAs de última generación de la marca Xilinx (serie Spartan-3).

En concreto se utilizan 4 piezas en encapsulado **BGA** (*Ball Grid Array*), el cual posee sus terminales de soldadura en forma de bolas de estaño-plomo ubicadas en la superficie inferior del IC, lo que provoca que la soldadura deje de estar visible.

Cada una de estas piezas tiene un número aproximado de 600 pines, que al estar situados bajo el encapsulado, **imposibilita la revisión de su correcto conexionado**, la ausencia de cruces o cortocircuitos, o incluso la posible rotura de alguna pista.

Además, a estas cuatro piezas esta conectada una CPLD, cuyo número de pines también es importante.

En total suman unos 3000 terminales, a los cuales están conectadas 1200 de las 1800 nets de la placa, lo que representa que **el 67% de las líneas de transmisión son inaccesibles de manera directa**.

Fig. 1.3 Cara TOP de la placa MAIN del NG25G

1.3.2 ¿Por qué es necesario realizar un test?

Imaginemos que una placa funciona de manera incorrecta.

Este mal funcionamiento puede ser provocado por:

1. Existencia de un fallo de montaje, automático (SMD) o manual (convencional).
   Suele ser la primera causa con diferencia, llegando al 90%.

2. Componente defectuoso (de fábrica o por destrucción involuntaria, como descargas, golpes..) .
   Mucho menos probable, pero en cualquier caso, se ha de localizar cual es el dispositivo defectuoso para proceder a su cambio.

3. Error de diseño.
   Ocurre en contadas ocasiones, y siempre después de haberse descartado las dos anteriores.

Si después de realizar una inspección visual no se detecta ningún fallo de montaje, podemos pensar que la anomalía está situada bajo cualquiera de las piezas BGA, lo cual, salvo contar con herramientas adecuadas y costosas, es imposible de verificar.

Como soluciones a este problema se podrían llevar a cabo las siguientes acciones.

- **Resoldar las piezas**

  De esta manera podríamos corregir defectos en la soldadura, pero no se detectarían fallos por problemas de pistas (roturas, cruces...).
  Existe la posibilidad de que el dispositivo sufra deterioros debido al recalentamiento.

- **Realizar un test**

  En el que se verifique la correcta conexión de los pines de cada dispositivo.
  Se detectarían todos los posibles fallos, y los dispositivos no sufrirían ningún riesgo.

Resoldar piezas es una tarea laboriosa, cuyo coste en mano de obra y material puede ser elevado, en cambio, un test realizado de manera automática, en cuestión de segundos, y sin suponer un riesgo para los dispositivos, parece una opción más que razonable.

1.4 Objetivos del test

El test debe cumplir los siguientes objetivos.

1. **Detectar errores de montaje en cuanto a:**

   - Mal contacto entre pistas del PCB y pines del dispositivo.
   - Cruce entre una o varias pistas/pines.
   - Cruce de alguna de las líneas a alimentación o masa.

2. **En caso de detectar fallo debe:**

   - Mostrar por pantalla y/o mediante fichero de texto, el dispositivo y pin/pista que produce el fallo.
   - Informar al usuario y/o mostrar información que le permitan al usuario "deducir" de que error de montaje se trata.

3. **Flexibilidad**

   Se piensa en un sistema que sea "transportable" (en un futuro) de una manera simple al resto de placas del grupo, amortizando de manera considerable, la inversión realizada en el proyecto.

4. **Duración**

   Pese a no existir más limite de tiempo que la paciencia del usuario, se debe procurar que el test no supere los dos-tres minutos de duración, pues, en el caso de que la dirección de la empresa opte por pasar el test a todas las placas, no repercuta de manera importante en el proceso de producción.

5. **Coste**

   Por tratarse de un proyecto que no genera beneficios económicos a la empresa, y una herramienta no imprescindible para la continuidad de esta, se pretende que la inversión en material y tiempo sean mínimos.

Considerando como ideal, que el test tuviera una eficiencia del 100% (siendo capaz de detectar un fallo en cualquiera de las 1200 líneas "testeables"), debido la interconexión de dispositivos no compatibles con el estándar JTAG, a un elevado número de nets con una única conexión a estos dispositivos, y a la imposibilidad (por destrucción o fallo funcional del módulo) de alterar los niveles lógicos de algunas nets, se establece bajo criterio del director y del autor, que una eficiencia cercana al **60-70%** cubriria las necesidades del proyecto.

CAPÍTULO 2. POSIBLES VIAS DE DESARROLLO

2.1 Análisis

En el anterior capítulo, hemos llegado a la conclusión de que el proyecto consiste en la realización de un test automático realizado sobre las piezas BGA del NG25G, que permita verificar el correcto conexionado de cada una de sus líneas de datos.

Dicho test, debería realizar una rutina que fuera conmutando los niveles lógicos de cada una de las líneas de transmisión de cada uno de los dispositivos, y comprobando que los pines del resto de dispositivos permanecen inmóviles, salvo aquellos que están conectados mediante pistas del PCB a las líneas bajo test.

Debemos observar cuales son las principales características de estos elementos, y buscar la manera con la que controlar los niveles lógicos de cada uno de los distintos dispositivos.

Tras un estudio sobre las piezas BGA, se observan dos posibles vías de desarrollo que permitirían la realización de dicho test.

2.2 Desarrollo de un firmware/software de test

Los dispositivos FPGA son elementos programables mediante lenguajes de alto nivel, que permiten el acceso y conexión interna entre todos y cada uno de los *puertos* (nombre único y distintivo de cada pin), realizando la función lógica deseada por el programador.
Por lo tanto, se plantea la posibilidad de programar una rutina de test que fuera conmutando el nivel lógico de cada uno de dichos puertos, y comprobando el correcto conexionado entre estos.

A continuación, comprobaremos si esta técnica se ajustaría a los objetivos y requisitos del test.

1. **Detección de errores de montaje.**

   Esta técnica permitiría la detección de todo tipo de anomalía en cuanto a conexionado se refiere, pues es posible “manejar” a nuestro antojo, cualquier puerto de manera independiente y detectar así cruces, cortocircuitos, etc.

El mayor inconveniente sería que previamente al test, es necesario cargar el software o rutina de test en cada una de las piezas, incrementando el tiempo necesario para realizar la comprobación de la placa.

2. Análisis y presentación de resultados

Como cada puerto tiene un nombre que lo identifica, es posible detectar el pin que falla, pero no es posible mostrar los resultados por pantalla, pues la PCB no dispone de ella.
Se podría desarrollar un software de conexión entre la placa y un PC, para que los resultados fueran volcados a este último, y mostrarlos por el monitor o en fichero de texto, pero incrementaríamos la complejidad del sistema.

3. Flexibilidad

Este és quizás el punto más crítico de esta técnica, pues la programación de la rutina de test sería especifica para cada placa, lo que conlleva los siguientes inconvenientes:

- Con el paso del tiempo, las PCB se van retocando en cuanto a diseño, añadiendo prestaciones, adaptándose a nuevas tecnologías o dispositivos, a nuevas normativas...
  Cada una de estas revisiones provocaría la revisión parcial o total de la rutina de test, apareciendo multitud de versiones de test para cada PCB.

- Uno de los objetivos del proyecto es que sea flexible y que en un futuro se pueda aplicar al resto de placas de la compañía. La elaboración de un software exclusivo para cada placa convertiría en múltiples proyectos testear otras placas, cuyo coste en tiempo y mano de obre sería elevado.

4. Duración

Esta es posiblemente la mayor ventaja de esta técnica.
La enorme capacidad y velocidad de calculo de los dispositivos permitiría una duración de test de apenas unos segundos.

5. Coste

El coste de este proyecto seria mínimo, pues el material necesario seria un PC para la programación de la rutina, y el tiempo necesario para el desarrollo de esta.
Sin embargo, debemos tener en cuenta que solo sería aplicable a un tipo y modelo de PCB, y en el caso de querer testear toda una gama de placas, el coste se multiplicaría.

2.3 Utilización del estándar JTAG

Una de las prestaciones que poseen estos dispositivos, es llevar incorporados dentro de sus funciones, el estándar *IEEE 1149.1 boundary-scan test.*

En 1990 aparece un nuevo estándar (1149.1 de la IEEE), que describe un bus de 4/5 líneas (TCK, TMS, TDI, TDO, TRST) capaz de controlar todas las funciones de un dispositivo.

Mediante este bus se puede fijar, cambiar, o capturar los estados lógicos de los puertos de las BGAs, sin afectar a su programación, por lo tanto, es posible realizar las mismas acciones que con la implementación de un software, pero sin la necesidad de programar los dispositivos.

A continuación, comprobaremos si esta técnica se ajustaría a los objetivos y requisitos del test.

1. **Detección de errores de montaje.**

   Esta técnica permitiría la detección de todo tipo de anomalía en cuanto a conexionado se refiere, pues es posible “manejar” a nuestro antojo cualquier puerto de manera independiente y detectar cruces, cortocircuitos...

   Como ventaja respecto al firmware/software , hay que destacar que no es necesaria una programación previa al test.

2. **Análisis y muestra de resultados.**

   El bus JTAG se controla mediante un PC conectado a la PCB.
   Esta conexión permitiría mostrar por pantalla y en tiempo real, el resultado del test.

3. **Flexibilidad**

   JTAG permite fijar y capturar niveles lógicos de cualquier tipo de dispositivo, independientemente de cual sea su conexión con el resto de dispositivos, interpretando los comandos o instrucciones suministradas por un software o fichero de control.

   Por lo tanto, y como ventaja más importante, JTAG es una técnica que permitiría testear cualquier PCB que contase con dispositivos compatibles con dicho estándar, situación que si se produce en la mayoría de PCB del grupo Trend Communications.

4. Duración

La duración del test dependerá en mayor medida de la velocidad de ejecución de las instrucciones del bus JTAG, la cual, depende del dispositivo de control JTAG y del puerto del PC al que esta conectado.
En el peor de los casos, se estima que el tiempo de test del 100% de la PCB es inferior a los 3 minutos, disminuyendo de manera exponencial en función del porcentaje de placa no testeada.

5. Coste

El coste del proyecto escogiendo la vía del JTAG, depende principalmente del dispositivo de control JTAG.
Este dispositivo se encarga de convertir las instrucciones enviadas por el PC (puerto USB o LPT1), en secuencias del bus JTAG (TMS, TCK, TDI), y de convertir las señales retornadas por la cadena JTAG (TDO) al PC.

En el caso de utilizar el estándar JTAG, se dispone de dos opciones en cuanto al dispositivo de control.

2.4 Opciones JTAG

2.4.1 Adquisición de herramienta comercial JTAG

JTAG Technologies (www.jtag.com) és una empresa especializada en la comercialización de dispositivos de control y test basados en dicho estándar.

Pese a ofrecer una gama de productos capaces de realizar las funciones necesarias para este proyecto, el elevado coste de estas herramientas, descarta su posible adquisición.

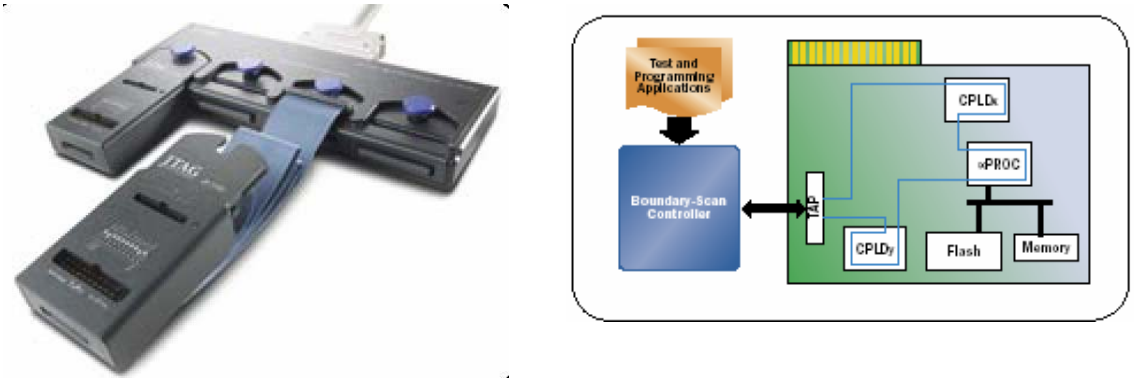

Fig. 2.1 Herramienta comercial para Test y control JTAG

2.4.2 Implementación de software y utilización del XPC3

Trend Communications dispone de un dispositivo de control JTAG denominado XPC3 (Xilinx parallel cable) utilizado normalmente para la programación de CPLD (*Complex Programmable Logic Device*).

Este dispositivo se conecta al puerto paralelo del PC, y al conector de 7 pines del PCB.
Se controla mediante un software suministrado con el propio dispositivo, y que en principio no serviría para nuestro proyecto.

Sin embargo, Trend dispone de una librería de funciones de control del XPC3 desarrollado por uno de los ingenieros del grupo, las cuales se podrían utilizar para el desarrollo de un nuevo software que si permitiese la ejecución del test.

Se trataría por tanto de desarrollar un software que interpretase las instrucciones procedentes de un fichero de entrada, y utilizase las funciones de dicha librería para ejecutarlas sobre el XPC3.

En ese caso, el coste en material seria nulo, y el único gasto seria el tiempo necesario para el desarrollo del software de control de XPC3 y el software de generación de las instrucciones de test.

2.5 Elección SOFTWARE + XPC3

Tras analizar las posibles vías, se opta por el desarrollo de un software de control JTAG, un software de generación de instrucciones de test, y la utilización del XPC3, teniendo en cuenta que:

- Pese a que ambas vías permiten la detección de fallos, el análisis de resultados, un coste mínimo, y una duración del test corta... la flexibilidad del JTAG frente a la nula del firmware/software, hace que descartemos esta última opción.

- El coste elevado de la herramienta comercial, frente a la mínima inversión necesaria en material escogiendo la opción del XPC3, hace que descartemos la primera opción.

CAPÍTULO 3. DESARROLLO DE LAS APLICACIONES

Una vez realizado un análisis sobre la placa y estudiado el funcionamiento del estándar JTAG, se decide desarrollar dos aplicaciones independientes entre si, que realicen las siguientes funciones.

- JTEST-A

Esta primera aplicación se encarga de recoger la información de dos ficheros de entrada, y generar un fichero de texto, cuyo contenido es una serie de instrucciones (en adelante *vectores de test*), para su posterior interpretación y ejecución con la aplicación JTEST-B.

El primero de estos ficheros, contiene información sobre el conexionado de la placa, como dispositivos, pines, y tipo de cada net.
Normalmente estos ficheros se pueden crear de manera automática con los programas utilizados para el diseño de los PCBs.

El segundo contiene el orden de la cadena JTAG en el PCB, además de información adicional sobre elementos de la placa, utilizados para para elevar el porcentaje de PCB testeado.

- JTEST-B

Esta aplicación se encarga de recoger los vectores de test del fichero de texto, y ejecutarlos a través del XPC3.

Realiza el control del bus JTAG, la programación de los dispositivos, y la comprobación de la correcta estrucura de la cadena.

Además, muestra por pantalla la evolución del test, y genera un fichero de texto con los resultados obtenidos.

Esta aplicación se ejecuta bajo una máquina Linux, no és autónoma, y como veremos más adelante, utiliza funciones y librerias procedentes de otro software.

3.1 Aplicación JTEST-A

Su principal objetivo es generar los vectores de test necesarios para la comprobación de cada una de las nets.

Para ello son necesarias cinco rutinas, realizando las siguientes funciones:

- La primera de ellas (denominada integra_converter), se encarga de filtrar y convertir a un formato más simple (fichero nets.txt), el contenido del fichero de netlist (fichero que contiene el conexionado entre los dispositivos de la placa).

- La segunda (jchain), recoge y almacena en memoria la información del fichero de configuración, para posteriormente cargar en memoria el contenido del fichero nets.txt.

- La tercera rutina (load_pinlist), detecta las nets testeables, y las configura y marca como tales.
  Además, en el caso de que el fichero de configuración contenga el pinlist, se encarga de actualizar los valores introducidos manualmente.

- La cuarta rutina (save_to_file) crea un nuevo fichero de configuración, añadiendo el pinlist de la placa.
  Este pinlist se puede modificar de manera manual, y utilizarlo como fichero de configuración al ejecutar de nuevo la aplicación JTEST-A.

- La ultima rutina (genera_vectores) guarda en un fichero de tipo (name.vt) los valores iniciales de las nets testeables, así como los valores que van tomando en función de la net testeada (vectores de test).

3.1.1 Rutina Integra_converter

3.1.1.1 Descripción

Su función es la de realizar un primer filtrado sobre el fichero de netlist, y convertir dicha información a un formato más simple.

El fichero de netlist tiene el siguiente formato, y es generado de manera automática por el programa ***Integra***, utilizado para el diseño de placas en el laboratorio de desarrollo.

El fichero de netlist esta ordenado por el Net Name (2V5Ref, 3V3, 3V3_BUF...), y presenta en forma de columnas, los dispositivos y pines que forman dicha net (Component Pin), el tipo de net (Net Class), y las coordenadas físicas del componente (Coordinates).

Integra_converter utiliza la columna *Net Class* para filtrar e ignorar aquellas nets que son del tipo *PowerXXX*.
Estas nets corresponden a líneas de alimentación, las cuales el estándar JTAG no permite testear.

Project:c:\temporal\2g5.tc

#	Component Pin	Net Name	Net Class	Coordinates		Width
1	IC58.10	2V5Ref	Standard Net Class	155.69 mm	164.755 mm	0.125 mm
2	IC92.1	2V5Ref	Standard Net Class	202.508 mm	185.232 mm	0.125 mm
3	C1181.2	2V5Ref	Standard Net Class	200.66 mm	185.407 mm	0.125 mm
4	R902.2	2V5Ref	Standard Net Class	200.81 mm	183.007 mm	0.125 mm
5	R901.2	2V5Ref	Standard Net Class	202.057 mm	185.407 mm	0.125 mm
6	R567.2	2V7Ref	Standard Net Class	162.700 mm	164.782 mm	0.125 mm
7	R525.1	2V7Ref	Standard Net Class	154.381 mm	164.274 mm	0.125 mm
8	C1191.2	2V7Ref	Standard Net Class	201.155 mm	183.515 mm	0.125 mm
9	R880.1	2V7Ref	Standard Net Class	196.47 mm	185.407 mm	0.125 mm
10	IC89.1	2V7Ref	Standard Net Class	200.463 mm	185.105 mm	0.125 mm
11	IC84.3	2V7Ref	Standard Net Class	187.325 mm	175.768 mm	0.125 mm
12	HI3.1	3V3	PowerLIN	134.207 mm	93.131 mm	0.30 mm
13	HI4.4	3V3	PowerLIN	135.215 mm	83.852 mm	0.30 mm
14	HI4.1	3V3	PowerLIN	135.215 mm	87.852 mm	0.30 mm
15	HI3.4	3V3	PowerLIN	130.207 mm	93.131 mm	0.30 mm
16	R550.2	3V3	PowerLIN	157.467 mm	167.767 mm	0.30 mm
17	C599.1	3V3	PowerLIN	129.273 mm	90.55 mm	0.30 mm
18	C610.1	3V3	PowerLIN	137.922 mm	83.045 mm	0.30 mm
19	HI5.1	3V3	PowerLIN	142.208 mm	93.131 mm	0.30 mm
20	C612.1	3V3	PowerLIN	137.401 mm	90.55 mm	0.30 mm
21	HI5.4	3V3	PowerLIN	138.208 mm	93.131 mm	0.30 mm
22	R524.1	3V3	PowerLIN	154.254 mm	168.338 mm	0.30 mm
23	SW9.2	3V3	PowerLIN	148.336 mm	163.714 mm	0.30 mm
24	TP69.1	3V3	PowerLIN	142.24 mm	89.916 mm	0.30 mm
25	IC51.1	3V3	PowerLIN	135.693 mm	88.834 mm	0.30 mm
26	R457.1	3V3	PowerLIN	135.649 mm	90.55 mm	0.30 mm
27	C611.1	3V3	PowerLIN	138.176 mm	90.375 mm	0.30 mm
28	R922.1	3V3_BUF	PowerLIN	50.787 mm	177.80 mm	0.30 mm
29	R920.1	3V3_BUF	PowerLIN	50.787 mm	179.578 mm	0.30 mm
30	L4.1	3V3_BUF	PowerLIN	76.202 mm	184.594 mm	0.30 mm
31	R31.1	3V3_BUF	PowerLIN	47.828 mm	186.626 mm	0.30 mm
32	L1.1	3V3_BUF	PowerLIN	71.943 mm	183.07 mm	0.30 mm
33	TP20.1	3V3_BUF	PowerLIN	82.423 mm	186.436 mm	0.30 mm

Fig. 3.1 Formato del fichero de netlist

La única información necesaria para la generación de los vectores de test es la que contienen las columnas *Component Pin* y *Net Name*, por lo tanto, crea un fichero (nets.txt) que contiene únicamente estos datos.

```
NETNAME      PIN

2V5Ref       IC58.10
2V5Ref       IC92.1
2V5Ref       C1181.2
2V5Ref       R902.2
2V5Ref       R901.2
2V7Ref       R567.2
2V7Ref       R525.1
2V7Ref       C1191.2
2V7Ref       R880.1
2V7Ref       IC89.1
2V7Ref       IC84.3
#FT_BCKB0_N IC90.A9
#FT_BCKB0_N J23.4
#FT_BCKB0_N R898.1
#FT_BCKB0_P IC90.B9
#FT_BCKB0_P J23.3
#FT_BCKB0_P R898.2
#FT_BCKB1_N IC90.C9
.
.
```

Fig. 3.2 Formato y contenido del fichero *nets.txt*

3.1.1.2 Diagrama de Flujo

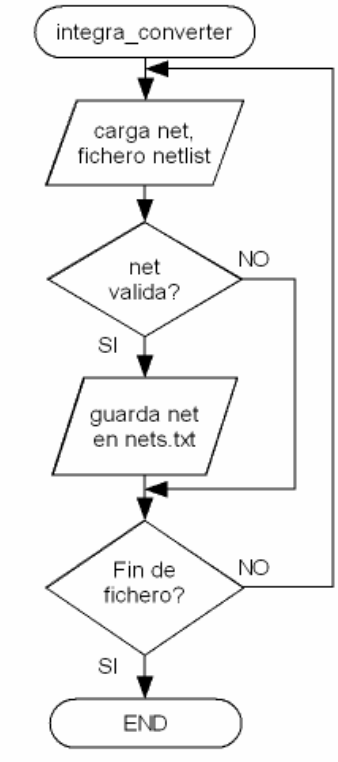

Fig. 3.3 Diagrama de flujo Integra_converter

3.1.1.3 Código fuente

Ver apartado 1.3 del Anexo A.

3.1.2 Rutina Jchain

3.1.2.1 Descripción

La rutina *Jchain* realiza las siguientes funciones:

1. Almacena en memoria el contenido del fichero de configuración, cuyo formato se muestra en la siguiente figura.

```
Nextgag     0.0
Device      ID    Specification
IC39        0     XC3S1500_FG676
IC40        1     XC3S1500_FG676
IC64        2     XC3S1500_FG676
IC63        3     XC3S1500_FG676
IC90        4     XC2C250_FG456
ENDDevice

Ignores
J16   J19   J10   J13   J9    J17   R903  R904  R390  R388
R372  R379  R334  R401  R472  R471  R470
ENDIgnores

Bypass
J15 1 3   J15 5 7   J15 9 11   J15 13 15   J15 17 19
J15 53 55   J15 57 59
ENDBypass

Pinlist
NET   CONEX DEV   IC-COMP     PIN   FUNC  VALUE NETNAME
0     0     4     IC90        A9    IN    X     #FT_BCKB0_N
0     1     -     J23         4     IN    X     #FT_BCKB0_N
0     2     -     R898        1     IN    X     #FT_BCKB0_N
1     0     4     IC90        B9    IN    X     #FT_BCKB0_P
1     1     -     J23         3     IN    X     #FT_BCKB0_P
1     2     -     R898        2     IN    X     #FT_BCKB0_P
2     0     4     IC90        C9    IN    X     #FT_BCKB1_N
2     1     -     J3          29    IN    X     #FT_BCKB1_N
2     2     -     R896        1     IN    X     #FT_BCKB1_N
3     0     4     IC90        D9    IN    X     #FT_BCKB1_P
3     1     -     J3          30    IN    X     #FT_BCKB1_P
3     2     -     R896        2     IN    X     #FT_BCKB1_P
4     0     4     IC90        D8    IN    X     #FT_BCKB2_N
4     1     -     J23         24    IN    X     #FT_BCKB2_N
4     2     -     R906        1     IN    X     #FT_BCKB2_N
ENDPinList
```

Fig. 3.4 Fichero de configuración (*config.txt*)

Formato del fichero de configuración:

- La primera línea contiene el nombre de la placa, seguido de la versión.

- A continuación, y hasta la etiqueta *ENDDevice*, se describe la posición (columna *ID*) que ocupa cada dispositivo (*Device*) en la cadena JTAG.

- Entre las etiquetas *Ignores* y *ENDIgnores,* se listan aquellos elementos o dispositivos que no alteran los niveles lógicos de la net, como por ejemplo, conectores al aire, resistencias para líneas diferenciales... y que por lo tanto, se pueden ignorar.

- Entre las etiquetas *Bypass* y *ENDBypass*, se listan aquellos dispositivos (p.e. *J15 1 3*) junto con dos pines, que interconectan (sin afectar los niveles lógicos) dos nets entre si.
  Ejemplos podrían ser Drivers o Buffers para reconstruir señales, o en el ejemplo anterior (conectores al aire), se pueden cruzar pines entre si (mediante conectores de sexo contrario) para alcanzar un mayor porcentaje de test.

- Por último, entre las etiquetas *Pinlist* y *ENDPinlist*, se lista el contenido en memoria referente a las conexiones de cada net.
  Este listado se genera una vez ejecutada la aplicación JTEST-A y cargado el fichero de netlist, por lo tanto, el primer fichero de configuración (lo debe crear el usuario), no contendrá esta información.
  Una vez ejecutada la aplicación JTEST-A, no se machaca el fichero de configuración entrado por el usuario, sino, que se crea uno nuevo cuyo nombre será del tipo nombre_placa.txt (p.e. Nextgag.txt), y que puede utilizarse como nuevo fichero de configuración.

Formato del Pinlist:

Cada net se identifica con un número entero (*NET*), y esta formada por numerosas conexiones (*CONEX*), identificadas también por números enteros.
Cada una de las conexiones tiene como mínimo un dispositivo JTAG (*DEV*), identificado con un número entero que corresponde a la posición dentro de la cadena JTAG, en caso de no ser un dispositivo JTAG, se marca con el símbolo "-".
El *PIN* es la patilla del componente al que esta conectada la net, identificada con la columna *NETNAME*.
Cuando el pin/puerto de un dispositivo se desconoce, se configura como (*FUNC*) "IN" y valor (*VALUE*) "X", si es conocido, toma los valores correspondientes, pudiendo ser "OUT" o "IN", y valor "1" o "0".
Estos dos últimos campos puede modificarlos el usuario manualmente, quedando actualizados para próximas ejecuciones de JTEST-A.

2. Carga en memoria las nets, dispositivos, y pines del fichero *nets.txt*, creado con la rutina anterior.

   Durante la carga, se tiene en cuenta el listado de dispositivos del tipo *Ignores*, los cuales no se almacenan en memoria.

3. Por último, en el caso de que el fichero de configuración contenga el Pinlist, y este hubiera sido modificado manualmente por el usuario, se actualiza el contenido en memoria con los nuevos valores.

3.1.2.2 Diagrama de flujo

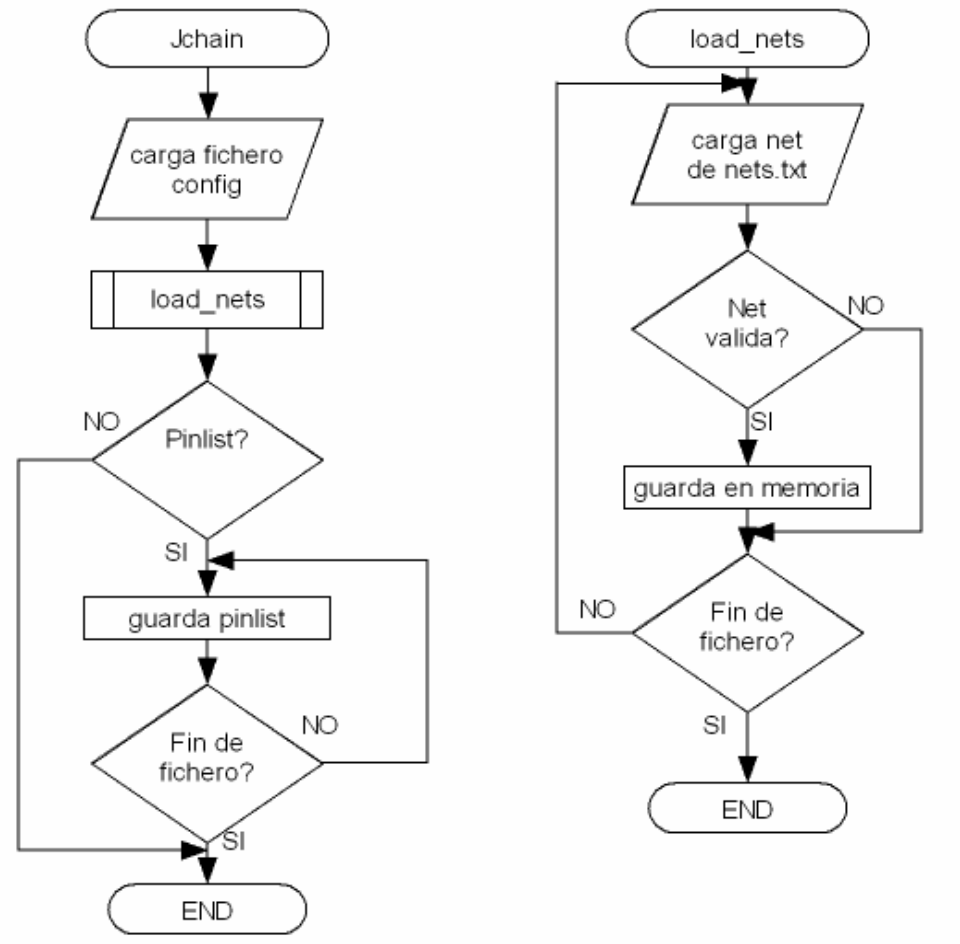

Fig. 3.5 Diagrama de flujo de la rutina *Jchain*

3.1.2.3 Código fuente

Ver apartado 1.4.1 y 1.4.2 del Anexo A.

3.1.3 Rutina Load_pinlist

3.1.3.1 Descripción

Load_pinlist se encarga principalmente de detectar las nets testeables y marcarlas como tales.

Por cuestiones de seguridad, y para no correr el riesgo de destruir algún dispositivo, someteremos a test solo aquellas nets que únicamente contienen dispositivos JTAG.

El usuario puede, mediante la combinación de los elementos *Ignores* y *Bypass*, conseguir un porcentaje elevado de nets testeables.
Para aquellas que no cumplan los requisitos pero el usuario tenga plena confianza en que no se va a destruir ningún elemento, es posible modificar los valores en el Pinlist del fichero de configuración.

La rutina load_pinlist realiza las siguientes funciones:

1. Actualiza los valores introducidos manualmente.

   Si el usuario a modificado los valores de las columnas "*FUNC*" y "*VALUE*", load_pinlist actualiza dichos valores, para tenerlos en cuenta a la hora de generar los vectores de test.
   Para una correcta actualización de estos valores, se establece una prioridad, de manera que los valores del campo "*VALUE*" de cada net, se actualizan al valor de dicho campo en la conexión (*CONEX*) cuyo índice es 0.

2. Ejecuta simplificación mediante elementos *Bypass*.

   Para un elemento tipo *Bypass* (p.e. J15 1 3), load_pinlist localiza aquellas nets donde aparece el dispositivo J15, y el pin/port 1 o 3.
   Escoge una de ellas, y añade los elementos que formaban la otra net.
   A continuación, elimina la segunda net.

   De esta manera, se combinan dos nets para formar una nueva con mas dispositivos, generalmente JTAG, aumentando el número de nets testeables.

3. Por último, una vez aplicados los cambios manuales y los elementos *Ignores* y *Bypass*, para aquellas nets que únicamente contienen dispositivos JTAG (testeables) se configura uno de los puertos como "OUT, 0" (*FUNC, VALUE*) y el resto como "IN, 0".

   Para la generación de los vectores de test, únicamente se tendrán en cuenta aquellas nets cuyo campo *VALUE* sea "0" o "1".

3.1.3.2 Diagrama de flujo

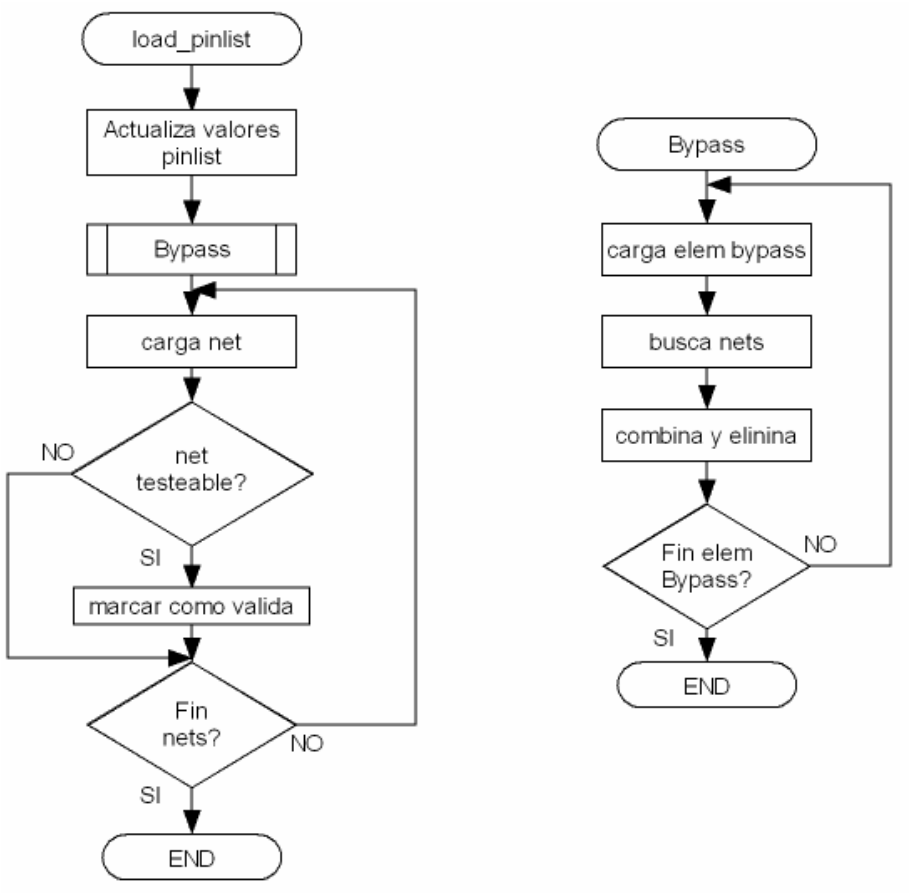

Fig. 3.6 Diagrama de flujo de la rutina *load_pinlist*

3.1.3.3 Código fuente

Ver apartado 1.5 del Anexo A.

3.1.4 Rutina Save_to_file

3.1.4.1 Descripción

La rutina Save_to_file guarda en un fichero de texto (con el mismo formato que el fichero de configuración), la información actualizada de los dispositivos JTAG, elementos Ignores, Bypass, y el pinlist generado con los datos anteriores.

Este fichero puede ser modificado ilimitadas veces por el usuario, y utilizarlo como nuevo fichero de configuración, para conseguir el porcentaje de test deseado.

3.1.4.2 Diagrama de flujo

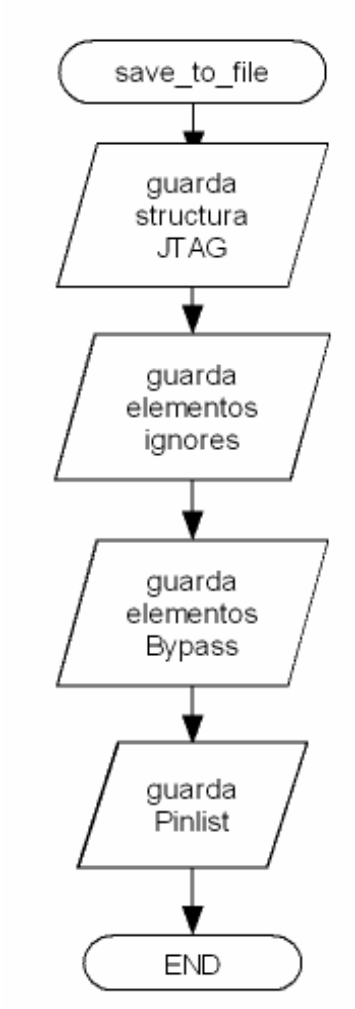

Fig. 3.7 Diagrama de flujo de la rutina save_to_file

3.1.4.3 Código fuente

Ver apartado 1.6 del Anexo A.

3.1.5 Rutina Genera_vectores

3.1.5.1 Descripción

Esta última rutina crea el fichero de vectores de test que será utilizado como fichero de entrada para la aplicación JETST-B.

```
Nextgag     0.0
Device      ID    Specification
IC39        0     XC3S1500_FG676
IC40        1     XC3S1500_FG676
IC64        2     XC3S1500_FG676
IC63        3     XC3S1500_FG676
IC90        4     XC2C250_FG456
ENDDevice

Initial_OUTs
3 F10 OUT 0 3 F11 OUT 0 3 F12 OUT 0 3 G12 OUT 0 3 G13 OUT 0
3 G9 OUT 0  3 G11 OUT 0 3 H13 OUT 0 3 B4 OUT 0  3 B6 OUT 0
ENDOuts

Initial_INPs
2 AA7 IN 0  2 AA8 IN 0  2 AA9 IN 0  2 AA10 IN 0 2 AA11 IN 0
2 Y8 IN 0   2 Y9 IN 0   2 Y11 IN 0  2 AF4 IN 0  2 AF6 IN 0
ENDInps

Test_vector
VT 0 #FT_BCKB0_N  EVT
VT 1 #FT_BCKB0_P  EVT
VT 2 #FT_BCKB1_N  EVT
VT 3 #FT_BCKB1_P  EVT
VT 240 ALNKC_0    3 F10 OUT 1 2 AA7 IN 1  EVT
VT 241 ALNKC_1    3 F11 OUT 1 2 AA8 IN 1  EVT
VT 242 ALNKC_2    3 F12 OUT 1 2 AA9 IN 1  EVT
VT 243 ALNKC_3    3 G12 OUT 1 2 AA10 IN 1 EVT
VT 244 ALNKC_4    3 G13 OUT 1 2 AA11 IN 1 EVT
VT 245 ALNKC_5    3 G9 OUT 1  2 Y8 IN 1   EVT
VT 246 ALNKC_6    3 G11 OUT 1 2 Y9 IN 1   EVT
VT 247 ALNKC_7    3 H13 OUT 1 2 Y11 IN 1  EVT
VT 261 ALNK_13    3 C5 OUT 1  2 AD5 IN 1  EVT
VT 814 LTHC_0     2 P4 OUT 1  1 N5 IN 1   EVT
VT 815 LTHC_1     2 P5 OUT 1  1 M5 IN 1   EVT
VT 816 LTHC_2     2 R5 OUT 1  1 L5 IN 1   EVT
ENDTest_Vector

Nets Testeadas 363 de 1113 (32%)
```

Fig. 3.8 Formato del fichero de vectores de test (*Nexgag.vt*)

1. La cabecera del fichero la componen el nombre y versión de placa, seguidos de la estructura JTAG.
   Estos datos permiten identificar la placa y serán de gran utilidad para ejecutar el fichero de test, únicamente sobre la placa que corresponde, eliminando el riesgo de destruir algún dispositivo.

2. A continuación se guardan los vectores iniciales de Ouput e Input.
   Estos vectores indican que niveles lógicos tienen los puertos cuando todavía no se a ejecutado ningún vector de test.
   Los puertos corresponden a aquellas nets consideradas como testeables, ignorando aquellas cuyos valores son desconocidos o poco fiables.

   De forma tabulada, se indica con cuatro campos, el estado en que debe quedar cada puerto.
   En el siguiente ejemplo, **3 F10 OUT 0**, el pin/port F10 del dispositivo 3, **será programado** a un valor lógico "**0**", mientras que en este otro **2 AA11 IN 0**, en el pin/port AA11 del dispositivo 2, debemos **leer** un nivel lógico "**0**".

3. Entre las etiquetas *Test_vector* y *ENDTest_vector,* y ordenados por filas, se guardan los vectores de test.

   Cada vector de test tiene el siguiente formato:

   VT 814 LTHC_0 2 P4 OUT 1 1 N5 IN 1 EVT

   Las etiquetas *VT* y *EVT*, indican el inicio y final de cada vector de test.
   El número entero (814) identifica dicho vector, y *LTHC_0*, corresponde a la net bajo test.
   A continuación, tabulado en columnas y con el mismo formato que los vectores iniciales, se listan los puertos que deben ser modificados respecto a los vectores iniciales.
   En este caso, el puerto P4 del dispositivo 2 debe quedar programado a un valor lógico "1", y en el puerto N5 del dispositivo 1, se debe leer un nivel lógico "1".
   Para aquellas net consideradas como no testeables, el vector de test únicamente contiene las etiquetas, y el número y nombre de la net.

4. Por último, y para tener una orientación sobre la efectividad del test, se muestra el numero de net testeadas sobre el total.

3.1.5.2 Diagrama de flujo

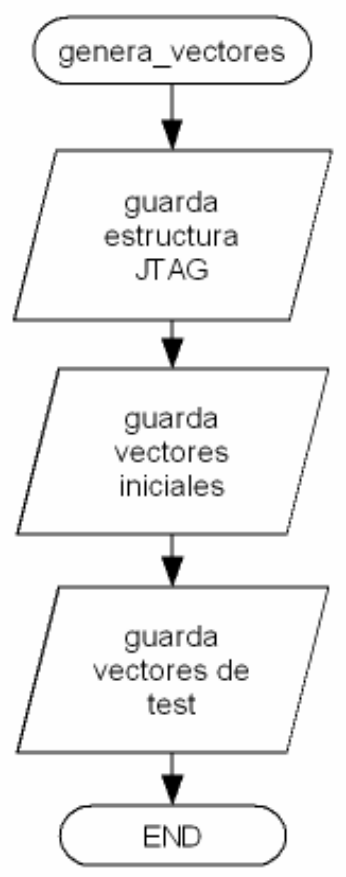

Fig. 3.9 Diagrama de flujo de la rutina *genera_vectores*

3.1.5.3 Código fuente

Ver apartado 1.7 del Anexo A.

3.2 Aplicación JTEST-B

JTEST-B es la aplicación encargada de la ejecución del test.
Se compone de una rutina que recorre el fichero de test, y una subrutina que carga y ejecuta los vectores de test.

No puede funcionar de forma autónoma, ya que utiliza funciones de una herramienta software diseñada por el laboratorio de desarrollo de Trend Communications.

Dicha herramienta se denomina JTD (*JTAG Debugger*), y se trata de un sistema operativo cuyos comandos permiten el control total de una cadena de dispositivos JTAG, a través del adaptador XPC3.

Por lo tanto, JTEST-B se ha diseñado y desarrollado para que forme parte de esta herramienta, convirtiéndose en uno de los posibles comandos dentro de JTD.

3.2.1 Entorno JTD

JTD es una aplicación (para linux) desarrollada por el ingeniero Jordi Colomer, que permite al usuario tener pleno control sobre una cadena de dispositivos JTAG.

Para ello, el usuario dispone de una serie de comandos que le permiten (entre otras funciones), autodetectar y listar los elementos que forman la cadena JTAG, fijar o leer el valor de los puertos de cualquiera de los dispositivos, cambiar las instrucciones o modos de funcionamiento, desplazar el contenido de los registros de datos, etc...

Fig. 3.10 Listado de comandos de la aplicación JTD

JTEST-B se ha diseñado para que quede integrado dentro del entorno JTD, escogiendose el comando “jt” para arrancar la aplicación.

Generalmente no se utilizan los comandos JTD directamente para la ejecución del test, pero si algunas de las funciones que utilizan los diferentes comandos.

Los comandos utilizados por JTEST-B son:

- Ad ➔ Necesario para seleccionar el adapter XPC3.
- Sn ➔ Autodotecta los dispositivos JTAG, y carga los ficheros BSDL correspondientes.
- Ex ➔ Ejecuta la instrucción *EXTEST*. Es necesario fijar este modo de trabajo para modificar los nieveles lógicos de los puertos.

Las funciones utilizadas por JTEST-B son:

- JTAG_Get_Name(&jch, i, name)➔ Función que devuelve el nombre del dispositivo seleccionado. Utilizada para comprobar la integridad de la cadena JTAG.

- JTAG_Shift(&jch) ➔ Guarda en el registro de salida, el contenido de la cadena JTAG, y carga en la cadena JTAG el contenido del registro de entrada.

- SetBitValue(str) ➔Carga en el registro de entrada, el contenido de la cadena str. La cadena STR contiene el nombre del puerto, y el valor a programar.

- ShowBitValue(str) ➔ Devuelve el valor lógico del puerto STR almacenado en el registro de salida.

3.2.2 Descripción

Una vez ejecutado el comando *jt* en la aplicación JTD, JTEST-B carga en memoria el contenido del fichero de test (p.e Nextgag.vt) y realiza la siguiente rutina.

1. Mediante el comando *sn* de JTD, lee la cadena JTAG de la placa, y la compara con la estructura descrita en la cabecera del fichero de test.
   Si el numero o los dispositivos son incorrectos, JTEST-B aborta el test y muestra por pantalla un mensaje de error.
   En caso contrario, continua la ejecución.

2. Crea e inicializa un fichero de resultados, y configura todos los dispositivos en modo EXTEST, para permitir la manipulación de todos sus puertos.

3. Carga en memoria los vectores iniciales de OUPUTS e INPUTS.

4. Se realiza una primera ejecución de la subrutina de test (sin vector de test) para compensar el desfase temporal.

5. A continuación, y para cada net (o vector de test) se realiza la siguiente subrutina de test.

   - Cargamos en el registro de entrada los valores iniciales de OUTs almacenados en memoria.

   - Modificamos en el registro de entrada los puertos de tipo OUT del vector de test

   - Modificamos el vector inicial de Inputs con los valores de los puertos de tipo IN del vector de test.

   - Ejecutamos la función JTAG_shift, para cargar en la cadena JTAG el registro de entrada.

   - Por último, y tendiendo en cuenta el desfase temporal (utilizando registros auxiliares), comparamos que los valores del vector inicial de inputs se corresponden con los valores leídos en el registro de salida.
     Si todo es correcto se da por valida la net. En caso contrario, se muestra por pantalla y se guarda en el fichero de resultados, aquellos puertos cuyo nivel lógico leído en el registro de salida no coincida con el nivel lógico del vector de inputs.

3.2.3 Diagrama de flujo

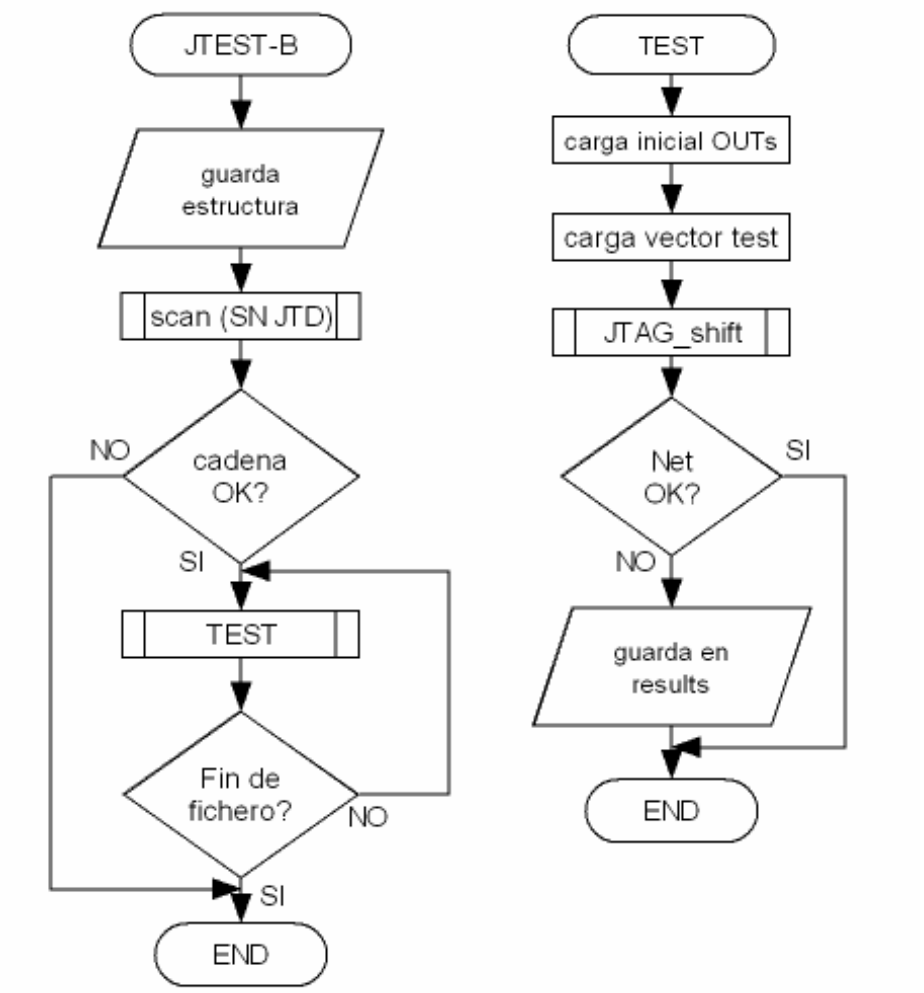

Fig. 3.11 Diagrama de flujo *JTEST-B*

3.2.4 Código fuente

Ver apartados 2.2 y 2.3 del Anexo A.

CAPÍTULO 4. VERIFICACIÓN DE FUNCIONAMIENTO

4.1Objetivo

El objetivo de estas pruebas es comprobar el correcto funcionamiento de las aplicaciones desarrolladas (cumpliéndose sus objetivos), la correcta integración de la aplicación JETST-B dentro del entorno JTD, y obtener una serie de parámetros (rendimiento, efectividad, duración) que nos permitan evaluar el éxito del proyecto.

Simularemos errores de montaje mediante la manipulación de uno de los conectores de la placa, y comprobaremos que el test es capaz de detectarlos, mostrarlos por pantalla, y dar información suficiente para prever cual a sido la causa de dicho fallo.

4.2 Equipamiento necesario

El equipamiento que utilizaremos para realizar las pruebas es el siguiente:

- PC con Windows XP, toma de red, y software para conexión SSH con una maquina linux.

- Maquina Linux (UPF) con toma de red, puerto paralelo (tipo LPT1) y aplicación JTEST-B y ficheros de test cargados en memoria.

- Utillaje de puesta en marcha para Vitoria Combo, compuesto por las placas mainframe, bus, y core. Esta ultima, debe tener el instalado el software de aplicación Nextgag.

- Placa MAIN y ADPOW del módulo Next Generation.
  Por motivos de seguridad, utilizaremos una placa main prototipo.
  Esta placa tiene la particularidad de que no posee 5 piezas FPGAs, sino solo 2, las necesarias para utilizar las funciones básicas.

- XPC3 (Xillinx parallel cable 3).

- Conector “mictor” hembra.

Fig. 4.1 Maquina Linux (UPF) con conector LPT1

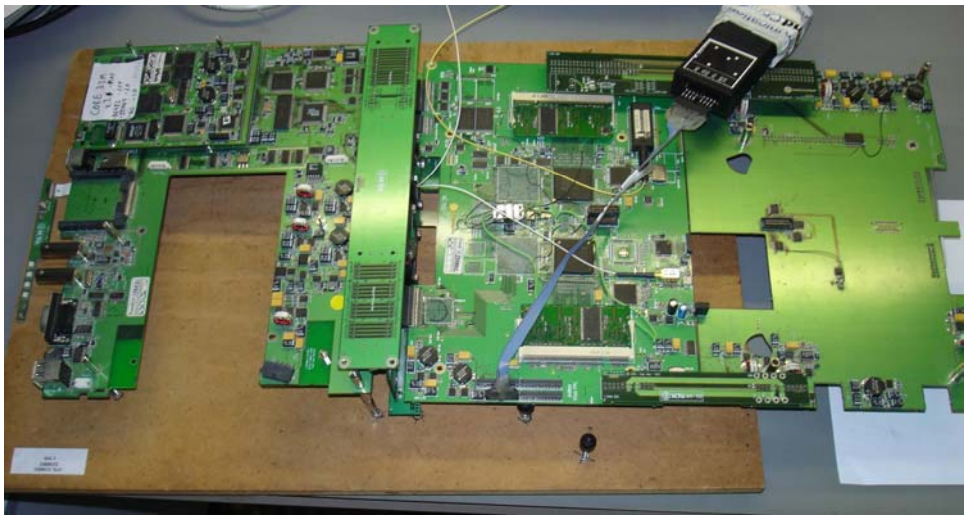

Fig. 4.2 Utillaje Victoria Combo, placas MAIN y ADPOW

Fig. 4.3 Xillinx Parallel cable III

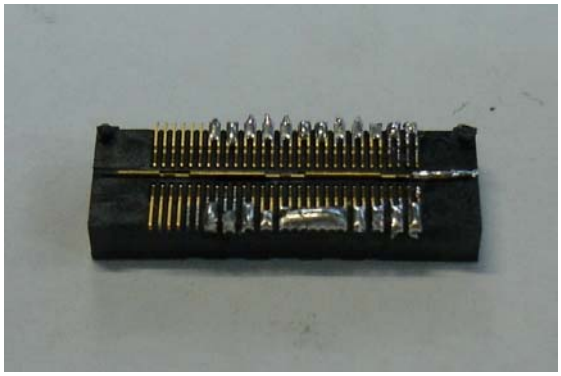

Fig. 4.4 Conector Hembra

4.3 Procedimiento

Conectaremos el XPC3 al puerto paralelo de la UPF, y arrancaremos el PC, la UPF y el utillaje de victoria combo.

A continuación abriremos una conexión SSH entre el PC y la UPF, y ejecutaremos el programa JTD (la versión que lleva implementada la aplicación JTEST-B).

Conectamos el otro extremo del XPC3 al conector J2 mostrado en la figura (Fig. 4.5), y ejecutaremos los siguiente comandos:

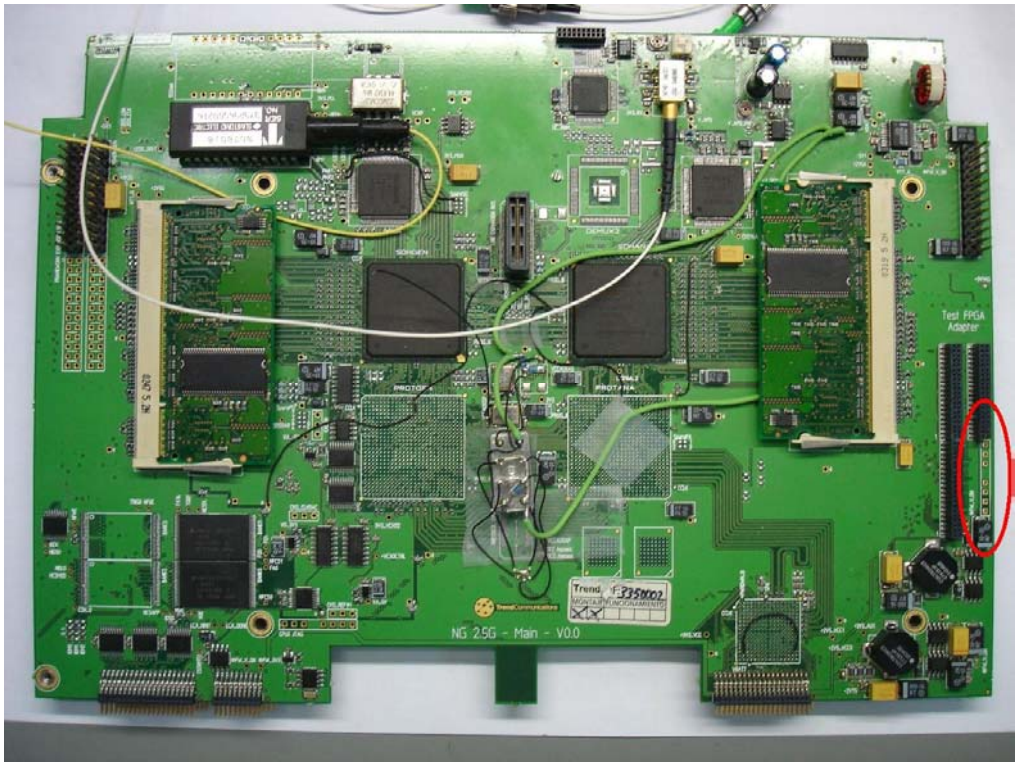

Fig. 4.5 Placa MAIN de pruebas

- Ad xpc3 ➔ Este comando selecciona el XPC3 como adapter .
- Jt ➔ Ejecuta la aplicación JTEST-B, mostrando por pantalla el resultado del test.

A partir de aquí, modificaremos las conexiones del conector mictor y ejecutando de nuevo el test, de manera que simulemos los errores de montaje más comunes, como son: cruces entre pistas, cortocircuitos a masa o alimentación, y cortes o mal contacto entre pistas y pines.

4.4 Pruebas de funcionamiento

4.4.1 Placa sin errores de montaje

En esta primera ejecución del test, el conector de pruebas esta modificado correctamente, simulando una placa sin errores de montaje.

Arrancamos el programa JTD, seleccionamos el adapter XPC3, y ejecutamos el test.

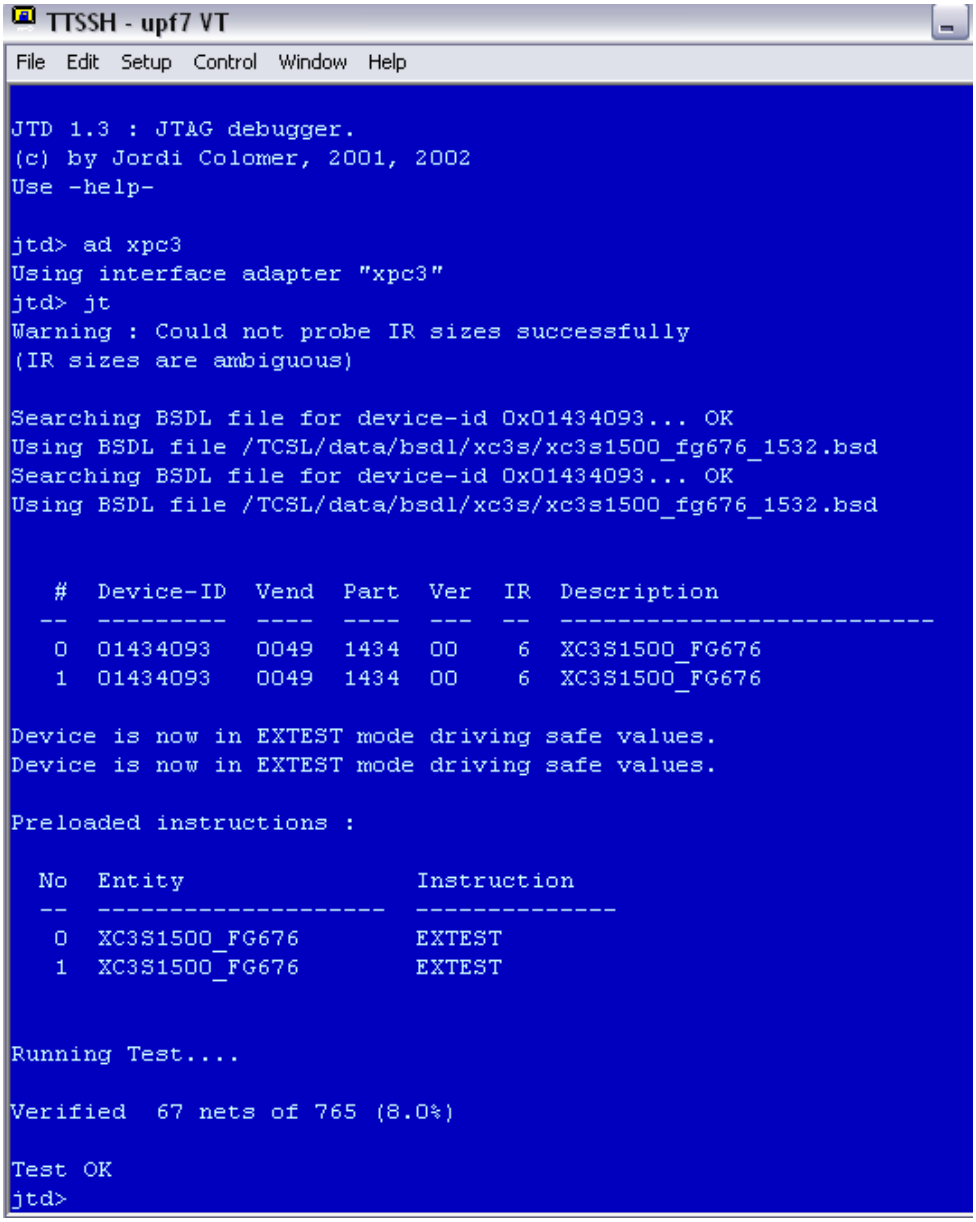

Fig. 4.6 Prueba de funcionamiento en una placa sin errores de montaje

Tras ejecutar el comando *JT*, JTEST-B detecta los dispositivos que forman la cadena JTAG (2 dispositivos del tipo XC3S1500_FG676), y carga en memoria los ficheros *BSDL (*Boundary-Scan Description Language*) para su control.

* Estos ficheros son suministrados por los fabricantes de los dispositivos, y se pueden descargar desde su pagina web.

A continuación, pone los dos dispositivos en modo *EXTEST* y configura sus puertos en modo seguro (*safe mode*).

Este modo consiste en colocar todos los puertos en modo input, y los terminales en alta impedancia.

Arranca el test, y tras 8 segundos de espera nos informa de que se han testeado 67 nets de un total de 765, y que el resultado a sido satisfactorio.

Si abrimos el fichero de resultados:

```
Nextgag   1.0
Device    ID   Specification
IC40      0    XC3S1500_FG676
IC64      1    XC3S1500_FG676
ENDDevice


Verified  67 nets of 765 (8.0%)

Test OK
```

Fig. 4.7 Fichero de resultados de una placa correctamente montada

Efectivamente, se ha realizado el test sobre dos dispositivos (los dos únicos montados) del tipo XC3S1500_FG676, que se ha testeado un 8% del total de nets, y que el resultado a sido OK.

4.4.2 Defecto de montaje tipo 1

Denominaremos error de montaje de tipo 1, a aquel que se produce cuando, a pesar de estar unidos por una pista del PCB, no existe conexión entre los pines de los dispositivos.
Generalmente se debe a cortes de pista, mala soldadura, vías defectuosas...

Para simular este defecto, modificaremos el conector dejando al aire una conexión entre pines que sí debería existir, y ejecutaremos de nuevo el comando *jt* para ver el resultado.

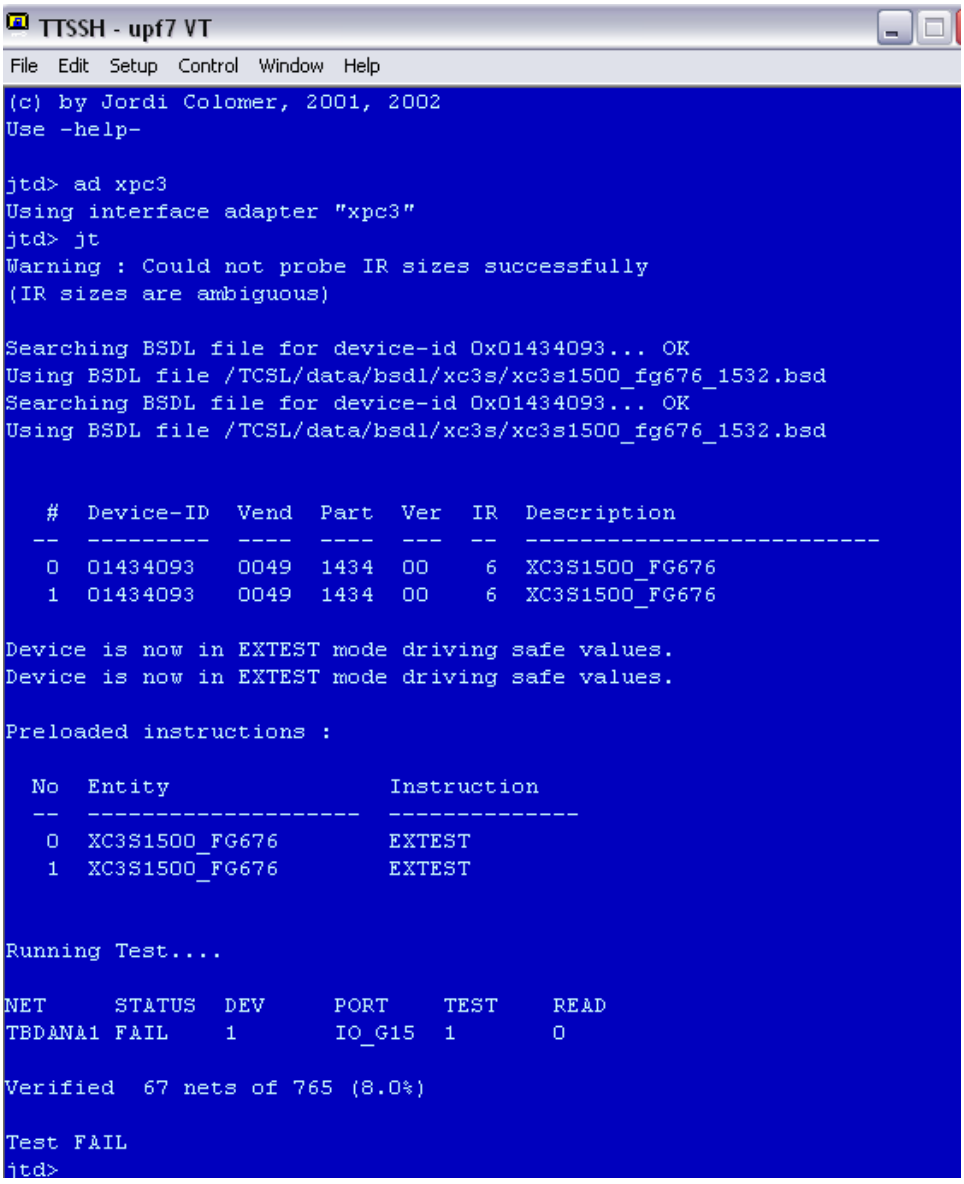

Fig. 4.8 Detección error de montaje tipo 1

En este caso, el resultado del test no es satisfactorio.

En primer lugar, JTEST-B nos indica que el error se ha detectado al realizar el test sobre la net TBDANA1, cuando en el port IO_G15 (pin G15) del dispositivo 1 (IC64), se esperaba leer un nivel lógico “1”, y se ha leído un “0”.

El nivel “1” programado a un extremo de la net, no a llegado al otro extremo, leyéndose por lo tanto, un valor nulo de tensión.

Hemos detectado por lo tanto, que existe un defecto de montaje, causado posiblemente por una mala conexión entre los pines de la net TBDANA1.

4.4.3 Defecto de montaje tipo 2

Definimos como error de montaje tipo 2, al que se produce cuando dos o mas líneas de datos se encuentran unidas de manera involuntaria.

Este error es muy común y se produce con frecuencia, ya que la distancia entre pistas y pines contiguos es mínima, y una excesiva cantidad de estaño puede provocar cruces entre ellos.

Se trata de uno de los errores de montaje más peligrosos, ya que puede llevar a la destrucción parcial o absoluta de los dispositivos.

Para simular este defecto de montaje, cortocircuitaremos dos pines del conector y ejecutaremos de nuevo el test.

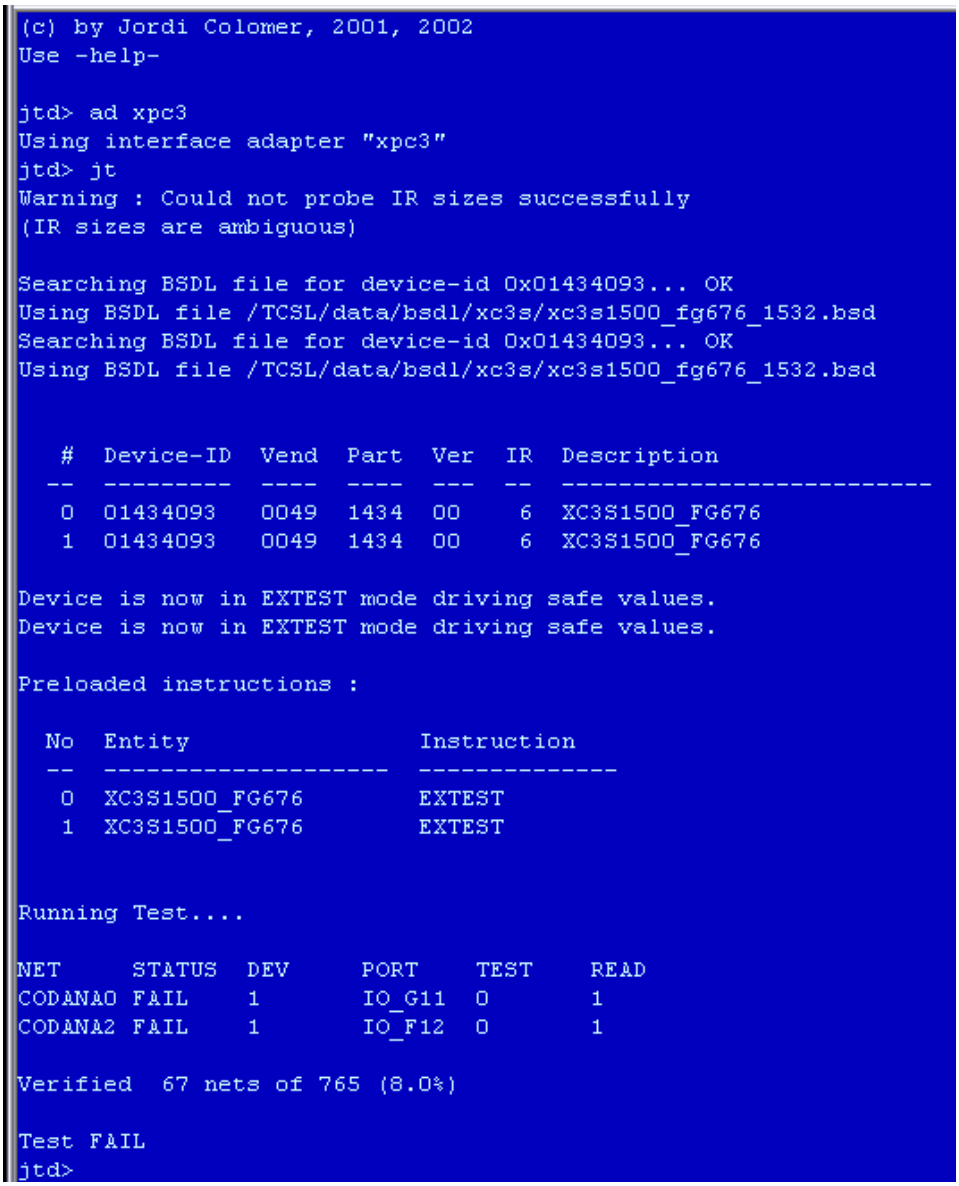

Fig. 4.9 Detección error de montaje tipo 2

En esta ocasión, JTEST-B nos retorna dos fallos.

El primero de ellos es en la net CODANA0, ya que en port IO_G11 se ha leido un valor 1, y se esperaba un 0.
El segundo tiene el mismo efecto, pero se ha producido en la net CODANA2, y el port IO_F12 del mismo dispositivo.

En un primer momento, podemos pensar que se produce una conexión a alimentación en ambos puertos, y que por esa razón el valor leído es un “1”.
Sin embargo, esa no es la causa.
Si se hubiera producido ese defecto, el numero de fallos seria mucho mayor, ya que cualquier test sobre cualquier net, nos retornaría un fallo.

Si consultamos el fichero de netlist, podemos comprobar rápidamente que el port IB_G11 del dispositivo 1, no pertenece a la net CODANA0, sino a la CODANA2, y que sucede lo mismo con el port IO_F12.

El hecho de que existan dos fallos, se debe a que al realizar el test sobre la net CODANA0, en el pin G11 (net CODANA2) se ha leído el valor 1, al realizar el test sobre la net CODANA2, se ha leído en el pin F12 (CODANA0) un 1.

Efectivamente, se deduce que se produce un cortocircuito entre ambas líneas, por lo tanto, se ha detectado el fallo, y su posible causa.

4.4.4 Defecto de montaje tipo 3

Denominamos error de montaje de tipo 3, al que se produce cuando existe una conexión involuntaria entre dos nets, siendo una de ellas estática, como ocurre con líneas de alimentación, masa, tensiones de referencia... etc.

Se trata de una variante particular del caso anterior, pero como veremos a continuación, su identificación es inmediata.

Para simular este defecto, uniremos una de las nets del conector, a una de las alimentaciones de la placa, y ejecutaremos de nuevo el test.

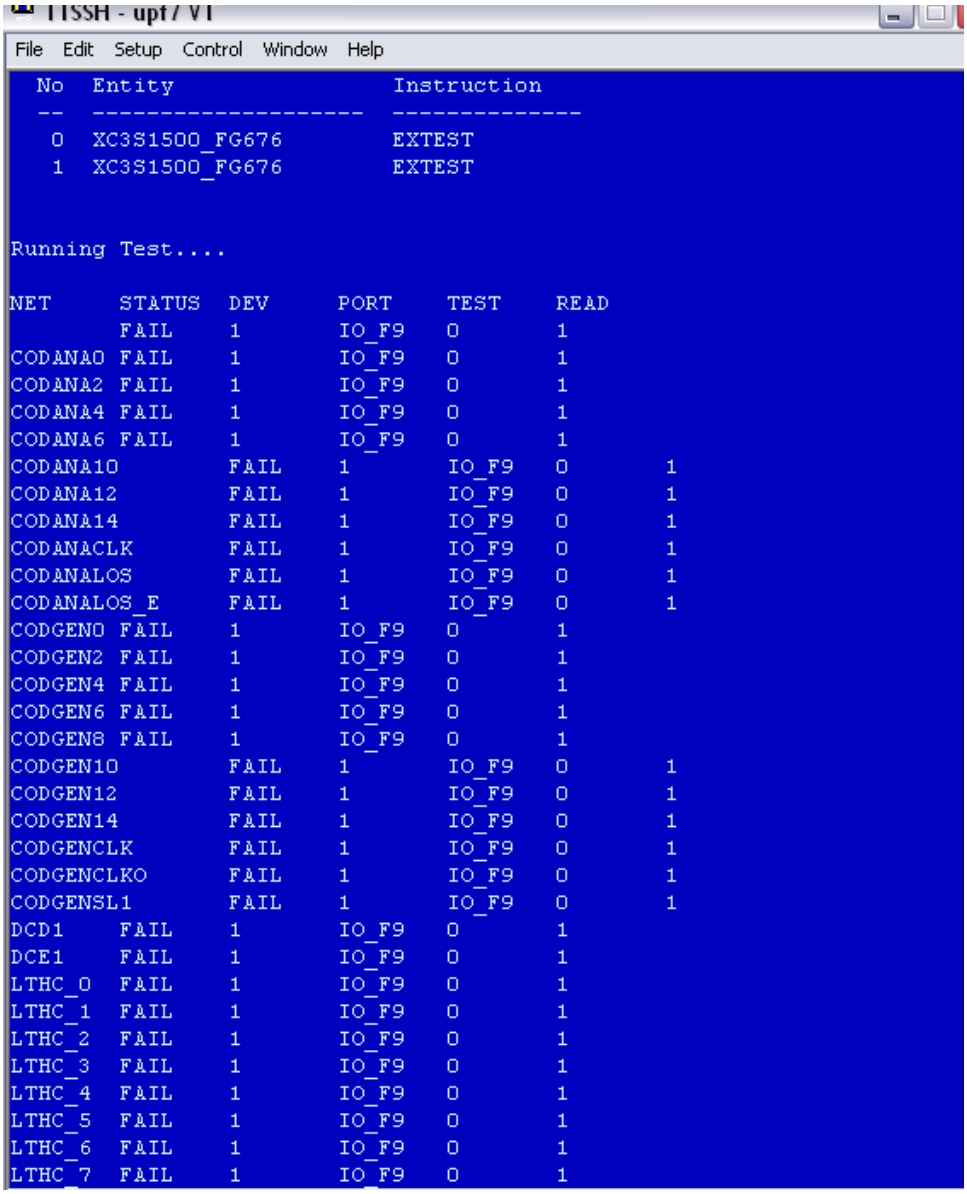

Fig. 4.10 Detección error de montaje tipo 3

Todas las nets testeadas (excepto la que contiene el puerto IO_F9 del dispositivo 1) han fallado en el mismo puerto y con los mismos síntomas.
En cada vector de test se ha detectado que el valor leído en dicho puerto se a mantenido constante a nivel “1”, cuando realmente debería tener un “0”, por lo tanto, la causa más probable es una derivación de la net CODANA8 a una de las alimentaciones de la placa, o línea estática a nivel “1”.

CAPÍTULO 5. CONCLUSIONES

5.1 Análisis de resultados

- En vista de los resultados, podemos asegurar que cualquier defecto de montaje que afecte a la conectividad de las FPGAs, y forme parte del conjunto de nets testeables, será detectado, mostrado por pantalla, y facilitada información suficiente para detectar su causa, cumpliéndose así, los dos primeros objetivos del TFC.

- Por cuestiones de seguridad, las pruebas se han realizado sobre una placa prototipo (versión 0.0) formada por 2 piezas BGA, testeandose un total de 67 nets sobre 765 (8%), en aproximadamente 8 segundos.

  Para la variante de 5 piezas (versión 1.0), este porcentaje alcanza el 32% (363 sobre 1113 nets), y mediante la utilización de conectores cruzados (similares al utilizado para las pruebas de funcionamiento) se espera alcanzar las 700 nets, rozando el 70% de probabilidad de detección de fallo.

- Ambas placas son diferentes en cuanto a versiones y dispositivos montados, sin embargo, el sistema se ejecuta correctamente sobre ambas, gracias a la flexibilidad que otorga la utilización de un fichero de configuración.
  Se cumple por tanto, el tercer objetivo del TFC.

- Con los datos obtenidos, podemos hacer una estimación del tiempo que necesita el sistema para testear el 70% de la placa.
  Debido al mayor número de dispositivos (5 dispositivos en lugar de 2), el tiempo necesario para desplazar los vectores de test será 5/2 mayor, mientras que el numero de vectores de test se multiplicará por 10 (700 frente a los 67).

  Por lo tanto, el tiempo necesario para realizar el test será de aproximadamente 200 segundos (algo mas de tres minutos).

  Pese a que supera en algunos segundos el tiempo marcado en los objetivos, se considera aceptable, y se intentará rebajar en futuras revisiones del sistema.

- Entre los presupuestos de Trend Communications, se encontraba la posible adquisición de una herramienta que realizase este tipo de test.

  Dicha herramienta tiene un coste aproximado de 24.000 euros, a los que hay que sumar las licencias del software que lo acompaña.

  Sin embargo, las necesidades de la empresa están muy por debajo de la capacidad de test de la herramienta, y se considera una inversión difícil de amortizar.

  La realización de este proyecto, permite a Trend Communications disponer de una herramienta de test que se ajuste a sus necesidades, utilizando recursos propios de la empresa, y realizando una mínima inversión de dinero.

5.2 Consideraciones y líneas futuras de diseño

- Durante la realización del proyecto, se ha observado que la gran limitación de este sistema radica en el número de dispositivos compatibles con el estándar JTAG.

  De aproximadamente 100 circuitos integrados (chips), solo 7 tienen implementado dicho estándar, en cambio, se ha conseguido testear el 40% de las líneas de datos.

  Si este número de chips se incrementase de manera considerable, o el estándar estuviera presente en aquellos dispositivos que concentran un mayor número de terminales, se podría aumentar esta cifra hasta el 80 o el 90%, rozando la cifra considerada como ideal para un sistema de diagnóstico.

  Por esta razón, se aconseja (siempre que sea posible) a los directores y responsables de los futuros proyectos de Trend Communications, la utilización de dispositivos que lleven incorporado dicho estándar, frente a aquellos similares que no lo implementen entre sus funciones.

- Durante el análisis de Victoria Combo, se ha observado que existe un bus que recorre todos los módulos conectados a la plataforma (Framework).

  Dicho bus utiliza dispositivos que llevan incorporados el estándar JTAG, y están presentes en todas las placas.

  Se platea la idea de unir dicho bus, al bus JTAG interno de la placa, lo que permitiría formar una cadena con más dispositivos JTAG, aumentando así el porcentaje de placa testeada.

- Ante el imparable aumento de la complejidad de las placas, el mayor número de dispositivos programables, y la cada vez mayor densidad de integración de estos dispositivos... se presume que el tiempo necesario para la ejecución del test sobre las placas que se desarrollaran en un futuro próximo puede ser elevado.

  Como se ha comentado durante en este documento, la velocidad del sistema depende principalmente del dispositivo de control JTAG, que en este caso se trata del XPC3.

  Este dispositivo utiliza el puerto paralelo del PC, cuya velocidad es relativamente baja frente a otros puertos mas modernos como pueden ser el USB o Firewire.

  Por lo tanto, se aconseja la compra de otro dispositivo que utilice estos puertos, permitiendo así la continuidad de las aplicaciones desarrolladas (JTD y JTEST).

CAPÍTULO 6. REFERENCIAS

Libros y obras consultadas

- Colomer, Jordi., “JTAG library” ., Barcelona ., 2002
- Oriol Giró, Jordi. “ Manual de fabricación del módulo NG25G”, Barcelona, 2003
- R G “Ben” Bennetts, “Boundary-Scan Tutorial”, vers. 2.1, septiembre 2002.

Webs de empresas dedicadas a la comercializacion de dispositivos de control JTAG.

- http://www.jtag.com
- http://www.xjtag.com
- http://www.goepel.com/content/html_en/index.php
- http://www.intellitech.com/

Tutoriales sobre el estándar *IEEE 1149.1 boundary-scan test*

- http://www.corelis.com/products/Boundary-Scan_Tutorial.htm
- http://focus.ti.com/lit/an/ssya002c/ssya002c.pdf
- http://www.asset-intertech.com/products/boundscan.htm

ANEXO A

Códigos fuente JTEST-A y JTEST-B

ANEXO A. Códigos fuente JTEST-A y JTEST-B

1. JTEST-A

1.1 Fichero de cabecera (*main.h*)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define N 20
#define M 5
#define C 3000
#define I 100

typedef struct
{
	int id;
	char name[N];		// ICXX
	char part_name[N];

}ic_t;

typedef struct
{
	char pcb_version[M];
	char pcb_name[N]; //name of PCB
	int numic; //total de ic
	ic_t chain[N]; // una chain esta formada por varios ic

}pcb_t;

typedef struct
{
	char dev[N];  // DEVICE
	char pin[M];   // PIN
	char function[M]; // IN, OUT, UNDEF
	char value[M]; // X, 0, 1
}conex_t;

typedef struct
{
	int numconex;
	char netname[N];
	conex_t conex[70];// CONEX type
}net_t;
```

```
typedef struct
{
	char pcb_name[N];
	char pcb_version[M];
	int numnets;  // num of nets
	net_t net[C]; // NET type

}nets_t;

typedef struct
{
	char device[N];

}ignores_t;

typedef struct
{
	int ignores_num;
	ignores_t ignores[C];
}iglist_t;

typedef struct
{
	char device[N];
	char pin1[N];
	char pin2[N];
}bypass_t;

typedef struct
{
	int bypass_num;
	bypass_t bypass[C];
}bylist_t;

void integra_converter();
void jchain(nets_t *nets,pcb_t *pcb,iglist_t *list,ic_t *ic,bylist_t *bypass);
void init_chain(pcb_t *pcb, iglist_t *list,bylist_t *bypass);
void ad_ic(pcb_t *pcb, ic_t *ic, int id, char name[N], char part_name[N]);
void print_chain(pcb_t *pcb);
void load_nets(nets_t *nets,pcb_t *pcb,iglist_t *list,ic_t *ic);
void init_nets(nets_t *nets);
int busca_net(nets_t *nets, char netname[N]);
void ad_net(nets_t *nets,char netname[N]);
void ad_conex(nets_t *nets,iglist_t *list,int pos,char dato[N]);
void valida_net(nets_t *nets,pcb_t *pcb);
void save_to_file(nets_t *nets,pcb_t *pcb,iglist_t *list,ic_t *ic,bylist_t *bypass);
int ignore_dev(iglist_t *list,char dev[N]);
void load_pinlist(nets_t *nets,pcb_t *pcb,bylist_t *bypass);
int is_JTAG(pcb_t *pcb,char name[N]);
void genera_vectores(nets_t *nets,pcb_t *pcb);
```

1.2. Fichero principal (*main.c*)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "main.h"

pcb_t pcb;
ic_t ic;
nets_t nets;
iglist_t list;
bylist_t bypass;

void main()
{

		printf("\n");
		printf("Test Vector Generator 1.0 (JTEST-A)\n");
		printf("(c) by Jose M. Barreiro, 2005\n");

		integra_converter();
		jchain(&nets,&pcb,&list,&ic,&bypass);
		load_pinlist(&nets,&pcb,&bypass);
		save_to_file(&nets,&pcb,&list,&ic,&bypass);
		genera_vectores(&nets,&pcb);
}
```

1.3 Código fuente de la rutina *Integra_converter*

```
void integra_converter()
{

	FILE *fe;
	FILE *fs;
	char dato[50];
	char dato1[N];
	char dato2[N];
	char dato3[N];
	char dato4[N];
	int id=0;
	int cont=1;
	char fichero_entrada[N];
	char fichero_salida[N];


	printf("\nIntroduce  fichero Netlist: ");
	scanf("%s",fichero_entrada);
```

```
	fe= fopen(fichero_entrada,"r");
	if(fe == NULL)

	{
		printf("\nError abriendo %s",fichero_entrada);
		exit(0);
	}

	fs= fopen("nets.txt","w");
	if(fs == NULL)
	{
		printf("\nError abriendo %s",fichero_salida);
		exit(0);
	}

	fprintf(fs,"NETNAME\t");
	fprintf(fs,"PIN\n");
	fscanf(fe,"%s", dato);
	while(!feof(fe))
	{
		fscanf(fe,"%s", dato1);
		id=atoi(dato1);

		if(id==cont)
		{
			fscanf(fe,"%s", dato2);
			if(strcmp(dato2,"mm"))
			{
			fscanf(fe, "%s", dato3);
			fscanf(fe, "%s", dato4);

			// Filtrado lineas de power.....
			if(strncmp(dato4,"Power",5) && strcmp(dato3,"...undef"))
				{
				fprintf(fs,"\n%s\t",dato3);
				fprintf(fs,"%s",dato2);
				}
			cont++;
			}
		}
	}
	fclose(fe);
	fclose(fs);
}
```

1.4 Código fuente de la rutina *Jchain*

1.4.1 Jchain

```
void jchain(nets_t *nets,pcb_t *pcb,iglist_t *list,ic_t *ic,bylist_t *bypass)
{
	FILE *fp;
	char dato[N];
	char name[N];
	char part_name[N];
	int id;
	char file[M];
	int net;
	int conex;
	char function[M];
	char value[M];


	// Inicializa estructuras pcb y list
	init_chain(pcb,list,bypass);

	printf("\nIntroduce el nombre del fichero de configurarion: ");
	scanf("%s",file);

	fp= fopen(file,"r");
	if(fp == NULL)

	{
		printf("\nError abriendo %s", file);
		exit(0);
	}

	// Guardamos identificador de placa
	fscanf(fp,"%s %s", name,dato);
	strcpy(pcb->pcb_name,name);
	strcpy(nets->pcb_name,name);
	strcpy(pcb->pcb_version,dato);
	strcpy(nets->pcb_version,dato);
```

```
// Guardamos la configuracion de la cadena JTAG
while(strcmp(dato,"ENDDevice") && !feof(fp))
        {
                fscanf(fp,"%s", dato);
                if(!strncmp(dato,"IC",2) )
                {
                        strcpy(name,dato);
                        fscanf(fp,"%s", dato);
                        id=atoi(dato);
                        fscanf(fp,"%s", dato);
                        strcpy(part_name,dato);
                        ad_ic(pcb, ic,id,name,part_name);
                }
        }

while(strcmp(dato,"Pinlist") && !feof(fp))
{

        fscanf(fp,"%s", dato);

        // Guardamos los elementos a ignorar
        if(!strcmp(dato,"Ignores"))
        {
        fscanf(fp,"%s", dato);
        while(strcmp(dato,"ENDIgnores"))
                        {
                        strcpy(list->ignores[list->ignores_num].device,dato);
                        list->ignores_num++;
                        fscanf(fp,"%s", dato);
                        }
        }

        // Guardamos los elementos BYPASS
        if(!strcmp(dato,"Bypass"))
        {
                fscanf(fp,"%s", dato);
                while(strcmp(dato,"ENDBypass"))
                {
                strcpy(bypass->bypass[bypass->bypass_num].device,dato);
                fscanf(fp,"%s", dato);
                strcpy(bypass->bypass[bypass->bypass_num].pin1,dato);
                fscanf(fp,"%s", dato);
                strcpy(bypass->bypass[bypass->bypass_num].pin2,dato);
                bypass->bypass_num++;
                fscanf(fp,"%s", dato);
                }
        }
}
```

```
// Cargamos las nets teniendo en cuenta la estructura JTAG y los IGNORES
load_nets(nets,pcb,list,ic);

// Si el fichero ya contiene el PinsList, lo cargamos en la estructura y actualizamos valores
	while(strcmp(dato,"NETNAME") && !feof(fp))
		{
		fscanf(fp,"%s", dato);
		}
	while(strcmp(dato,"ENDPinList") && !feof(fp))
		{
			fscanf(fp,"%s", dato);
			net=atoi(dato);
			fscanf(fp,"%s", dato);
			conex=atoi(dato);
			fscanf(fp,"%s", dato);
			fscanf(fp,"%s", dato);
			fscanf(fp,"%s", dato);
			fscanf(fp,"%s", function);
			fscanf(fp,"%s", value);
			strcpy(nets->net[net].conex[conex].function,function);
			strcpy(nets->net[net].conex[conex].value,value);
			fscanf(fp,"%s", dato);
		}
	fclose(fp);
}


void init_chain(pcb_t *pcb, iglist_t *list,bylist_t *bypass)

{
	// Inicializa estructuras PCB,LIST,BYPASS
	pcb->numic=0;
	list->ignores_num=0;
	bypass->bypass_num=0;
}

void ad_ic(pcb_t *pcb, ic_t *ic, int id, char name[N], char part_name[N])
{
	//Añadimos IC
	pcb->chain[pcb->numic].id=id;
	strcpy(pcb->chain[pcb->numic].name,name);
	strcpy(pcb->chain[pcb->numic].part_name,part_name);
	pcb->numic++;
}
```

1.4.2 Subrutina *load_nets*

```
void load_nets(nets_t *nets,pcb_t *pcb,iglist_t *list,ic_t *ic)
{
	FILE *fn;
	char dato[N];
	char dato_aux[N];
	int pos;
	int i=0;
	int j=0;

	//Inicia la estructura Nets
	init_nets(nets);

	fn= fopen("nets.txt","r");
	if(fn == NULL)

	{
		printf("\nError abriendo nets.txt");
		exit(0);
	}
	//Ignoramos las dos primeras palabras
	fscanf(fn,"%s", dato);
	fscanf(fn,"%s", dato);

	//Cargamos valores en la estructura
	while(!feof(fn))
	{
		fscanf(fn,"%s", dato);

		// Si la net no es valida, no se almacena en la estructura
		if(nets->numnets==0)
		{
		strcpy(dato_aux,dato);
		}

		if(strcmp(dato,dato_aux))
		{
			valida_net(nets,pcb);
		}
		strcpy(dato_aux,dato);

		//retorna la posicion de la net, o -1 si no esta
		pos=busca_net(nets, dato);
```

```
                if(pos==-1)
                {
                // Creamos  nueva net en la estructura, y añadimos una conexión
                        ad_net(nets,dato);
                        fscanf(fn,"%s", dato);
                        ad_conex(nets,list,nets->numnets-1,dato);
                }
                else
                {
                        // Añadimos una conexion
                        fscanf(fn,"%s", dato);
                        ad_conex(nets,list,pos,dato);
                }
        }
        printf("Hay  %d nets\n", nets->numnets);
}

void init_nets(nets_t *nets)
{
        nets->numnets=0;
}

int busca_net(nets_t *nets, char netname[N])
{
        int i;
        int flag=-1; // retorna -1 si no encuentra la net, sino retorna la posicion

        for(i=0;i<nets->numnets;i++)
        {
                if(strcmp(nets->net[i].netname,netname)==0)
                        return (i);
        }
        return (flag);
}

void ad_net(nets_t *nets,char netname[N])
{
        //Añadimos una net
        strcpy(nets->net[nets->numnets].netname,netname);
        nets->net[nets->numnets].numconex=0;
        nets->numnets++;
}
```

```
void ad_conex(nets_t *nets,iglist_t *list,int pos,char dato[N])
{

	//Añadimos conexion nueva
	int i=0;
	int j=0;
	char dev[N];
	char pin[M];

	// Obtenemos el dispositivo
	while(dato[i]!='.')
	{
		dev[i]=dato[i];
		i++;
	}
	dev[i]='\0';
	i++;
	// Obtenemos el pin
	while(dato[i]!='\0')
	{
		pin[j]=dato[i];
		i++;
		j++;
	}
	pin[j]='\0';

	// Si no es un dispositivo a ignorar, añadimos todos los parametros
	if(!ignore_dev(list,dev))
	{
	strcpy(nets->net[pos].conex[nets->net[pos].numconex].dev,dev);
	strcpy(nets->net[pos].conex[nets->net[pos].numconex].pin,pin);
	strcpy(nets->net[pos].conex[nets->net[pos].numconex].function,"IN");
	strcpy(nets->net[pos].conex[nets->net[pos].numconex].value,"X");
	nets->net[pos].numconex++;
	}
}

void valida_net(nets_t *nets,pcb_t *pcb)
{
	int i;
	int j;
	int flag=0;		// Si flag es 1, dispositivo  JTAG, net valida
```

```
	for(i=0;i<nets->net[nets->numnets-1].numconex;i++)
	{
	for(j=0;j<pcb->numic;j++)
	{
	if(!strcmp(nets->net[nets->numnets-1].conex[i].dev,pcb->chain[j].name))
			{
				flag=1;
			}
		}
	}
	if(flag==0)
	{
	nets->numnets--;
	}

}

int ignore_dev(iglist_t *list,char dev[N])
{
	int i;
	int flag=0;   // Si flag es 1, es un dispositivo a ignorar, conexion no valida

	for(i=0;i<list->ignores_num;i++)
	{
		if(!strcmp(dev,list->ignores[i].device))
			flag=1;
	}

	return (flag);
}
```

1.5 Código fuente de la rutina *Load_pinlist*

```
void load_pinlist(nets_t *nets,pcb_t *pcb,bylist_t *bypass)
{
	int i,j,k;
	char value[M];
	int flag=0;
	int net1,net2;
	int conex1,conex2;

// Si se fuerza un valor a X, las conexiones restantes se configuran como X
for(i=0;i<nets->numnets;i++)
	{
	strcpy(value,nets->net[i].conex[0].value);
	for(j=1;j<nets->net[i].numconex;j++)
		{
		strcpy(nets->net[i].conex[j].value,value);
		}
	}
// Tenemos en cuenta los elementos tipo BYPASS

for(i=0;i<bypass->bypass_num;i++)
{
	for(j=0;j<nets->numnets;j++)
	{
		for(k=0;k<nets->net[j].numconex;k++)
		{
		if(!strcmp(bypass->bypass[i].device,nets->net[j].conex[k].dev)	&&
!strcmp(bypass->bypass[i].pin1, nets->net[j].conex[k].pin))
			{
				net1=j;
				conex1=k;
			}
		if(!strcmp(bypass->bypass[i].device,nets->net[j].conex[k].dev)	&&
!strcmp(bypass->bypass[i].pin2, nets->net[j].conex[k].pin))
			{
				net2=j;
				conex2=k;
			}
		}
	}
```

```
        for(k=0;k<nets->net[net1].numconex;k++)
        {
        if(is_JTAG(pcb,nets->net[net1].conex[k].dev))
        {
                strcpy(nets->net[net2].conex[conex2].dev,nets-
>net[net1].conex[k].dev);
                strcpy(nets->net[net2].conex[conex2].function,nets-
        >net[net1].conex[k].function);
                strcpy(nets->net[net2].conex[conex2].pin,nets-
>net[net1].conex[k].pin);
                strcpy(nets->net[net2].conex[conex2].value,nets-
>net[net1].conex[k].value);
                        }
                }
        for(k=0;k<nets->net[net2].numconex;k++)
        {
                if(is_JTAG(pcb,nets->net[net1].conex[k].dev) && (k!=conex2))
                {
                strcpy(nets->net[net1].conex[conex1].dev,nets-
>net[net2].conex[k].dev);
                strcpy(nets->net[net1].conex[conex1].function,nets-
>net[net2].conex[k].function);
                strcpy(nets->net[net1].conex[conex1].pin,nets-
>net[net2].conex[k].pin);
                strcpy(nets->net[net1].conex[conex1].value,nets-
>net[net2].conex[k].value);
                }
        }
        for(k=net2;k<nets->numnets-1;k++)
        {
                nets->net[k]=nets->net[k+1];
        }
        nets->numnets--;
}

// Para una NET compuesta únicamente por dispositivos JTAG, Configura uno
como OUT 0, y los demás como IN 0

        flag=0;
        for(i=0;i<nets->numnets;i++)
        {
                for(j=0;j<nets->net[i].numconex;j++)
                {
                        if(!is_JTAG(pcb,nets->net[i].conex[j].dev))
                                flag=1;
                }
                if(!flag && (nets->net[i].numconex>1))
                        {
                        strcpy(nets->net[i].conex[0].function, "OUT");
                        strcpy(nets->net[i].conex[0].value, "0");
```

```
                        for(j=1;j<nets->net[i].numconex;j++)
                                {
                                strcpy(nets->net[i].conex[j].function, "IN");
                                strcpy(nets->net[i].conex[j].value, "0");
                                }
                        }
                flag=0;
        }
}
```

1.6 Código fuente de la rutina *Save_to_file*

```
void save_to_file(nets_t *nets,pcb_t *pcb,iglist_t *list,ic_t *ic,bylist_t *bypass)
{
//Genera un fichero con toda la informacion obtenida
        int i;
        int j;
        int z;
        char id;
        char dev[N];
        char pin[M];
        char function[M];
        char value[M];
        char netname[N];
        char pcb_name[N];

        FILE *fs;

        strcpy(pcb_name,pcb->pcb_name);
        strcat(pcb_name,".txt");

        fs= fopen(pcb_name,"w");
        if(fs == NULL)

        {
                printf("\nError abriendo chain.txt");
                exit(0);
        }

//La cabecera del fichero contiene la placa y la version
fprintf(fs,"%s\t%s",pcb->pcb_name,pcb->pcb_version);
fprintf(fs,"\nDevice\tID\tSpecification");
```

```
// Guardamos la configuracion de la cadena JTAG
        for(z=0;z<pcb->numic;z++)
        {
        fprintf(fs,"\n%s\t%d\t%s",pcb->chain[z].name,pcb->chain[z].id,pcb-
        >chain[z].part_name);
        }
        fprintf(fs,"\nENDDevice");

        // Guardamos los elementos a ignorar
        if(list->ignores_num>0)
        {
                fprintf(fs,"\n\nIgnores\n");

                for(i=1;i<=list->ignores_num;i++)
                {
                fprintf(fs,"%s\t",list->ignores[i-1].device);
                if(i%10==0) fprintf(fs,"\n");
                }

        fprintf(fs,"\nENDIgnores");
        }

// Guardamos los elementos Bypass
        if(bypass->bypass_num>0)
        {
                fprintf(fs,"\n\nBypass\n");

                for(i=1;i<=bypass->bypass_num;i++)
                {
                fprintf(fs,"%s  %s  %s       ",bypass->bypass[i-1].device,bypass-
>bypass[i-1].pin1,bypass->bypass[i-1].pin2);
                if(i%8==0) fprintf(fs,"\n");
                }

                fprintf(fs,"\nENDBypass");
        }
```

```
// Guardamos el PINList

        fprintf(fs,"\n\nPinlist");
        fprintf(fs,"\nNET\tCONEX\tDEV\tIC-
COMP\tPIN\tFUNC\tVALUE\tNETNAME\t");

        for(i=0;i<nets->numnets;i++)
        {
        for(j=0;j<nets->net[i].numconex;j++)
                {
                for(z=0;z<pcb->numic;z++)
                        {
                        if(!strcmp(nets->net[i].conex[j].dev,pcb->chain[z].name))
                                {
                                        id=z+48;
                                        break;
                                }
                                else id='-';
                        }
                        strcpy(dev,nets->net[i].conex[j].dev);
                        strcpy(pin,nets->net[i].conex[j].pin);
                        strcpy(function,nets->net[i].conex[j].function);
                        strcpy(value,nets->net[i].conex[j].value);
                        strcpy(netname,nets->net[i].netname);

        fprintf(fs,"\n%d\t%d\t%c\t%s\t%s\t%s\t%s\t%s",i,j,id,dev,pin,function,
value,netname);
                        }
                }
        fprintf(fs,"\nENDPinList");
}
```

1.7 Código fuente de la rutina *Genera_vectores*

```
void genera_vectores(nets_t *nets,pcb_t *pcb)
{
	int i;
	int j;
	int z;
	int cont=0;
	int flag=0;
	int por;
	char id;
	char pcb_name[N];
	char value[M];

	FILE *fp;

	strcpy(pcb_name,pcb->pcb_name);
	strcat(pcb_name,".vt");

	fp= fopen(pcb_name,"w");
	if(fp == NULL)

	{
		printf("\nError creando fichero");
		exit(0);
	}

//La cabecera del fichero contiene la placa y la version
	fprintf(fp,"%s\t%s",pcb->pcb_name,pcb->pcb_version);

	fprintf(fp,"\nDevice\tID\tSpecification");

 // Guardamos la configuracion de la cadena JTAG
	for(i=0;i<pcb->numic;i++)
	{
	fprintf(fp,"\n%s\t%d\t%s",pcb->chain[i].name, pcb->chain[i].id, pcb-
>chain[i].part_name);
	}
	fprintf(fp,"\nENDDevice");
```

```
// Generamos vector inicial de OUTs

fprintf(fp,"\n\nInitial_OUTs");
for(i=0;i<nets->numnets;i++)
{
	for(j=0;j<nets->net[i].numconex;j++)
	{
	if(is_JTAG(pcb,nets->net[i].conex[j].dev) && (!strcmp(nets->net[i].conex[j].function,"OUT")))
		{
		//Buscamos posicion en la cadena JTAG
			for(z=0;z<pcb->numic;z++)
			{
			if(!strcmp(nets->net[i].conex[j].dev,pcb->chain[z].name))
				{
					id=z+48;
					break;
				}
			}
		if(!(cont%5)) fprintf(fp,"\n");
		fprintf(fp,"%c %s %s %s\t",id, nets->net[i].conex[j].pin,nets->net[i].conex[j].function,nets->net[i].conex[j].value);
			cont++;
		}
	}
}
	fprintf(fp,"\nENDOuts");
	cont=0;

// Generamos vector inicial de INPs

	fprintf(fp,"\n\nInitial_INPs");

	for(i=0;i<nets->numnets;i++)
	{
		for(j=0;j<nets->net[i].numconex;j++)
		{
		if(is_JTAG(pcb,nets->net[i].conex[j].dev) && (!strcmp(nets->net[i].conex[j].function,"IN")) && (strcmp(nets->net[i].conex[j].value,"X")))
			{
			//Buscamos posicion en la cadena JTAG
			for(z=0;z<pcb->numic;z++)
				{
				if(!strcmp(nets->net[i].conex[j].dev,pcb->chain[z].name))
					{
						id=z+48;
						break;
					}
				}
```

```
                        if(!(cont%5)) fprintf(fp,"\n");
                        fprintf(fp,"%c %s %s %s\t",id, nets->net[i].conex[j].pin,nets-
>net[i].conex[j].function,nets->net[i].conex[j].value);
                                cont++;
                        }

                }

        }
        fprintf(fp,"\nENDInps");

// Generamos vectores

cont=0;
fprintf(fp,"\n\nTest_vector");
for(i=0;i<nets->numnets;i++)
{
        strcpy(value,"X");
        flag=0;
        fprintf(fp,"\nVT %d %s\t",i,nets->net[i].netname);
        for(j=0;j<nets->net[i].numconex;j++)
        {
                if(is_JTAG(pcb,nets->net[i].conex[j].dev)      &&      (!strcmp(nets-
>net[i].conex[j].function,"OUT")))
                {
                //Buscamos posicion en la cadena JTAG
                for(z=0;z<pcb->numic;z++)
                {
                if(!strcmp(nets->net[i].conex[j].dev,pcb->chain[z].name))
                                {
                                id=z+48;
                                break;
                                }
                }
                strcpy(value,"1");
                fprintf(fp,"%c  %s  %s  %s\t",id,  nets->net[i].conex[j].pin,nets-
>net[i].conex[j].function,value);
                        flag=1;
                }
        }
        for(j=0;j<nets->net[i].numconex;j++)
        {
        if(is_JTAG(pcb,nets->net[i].conex[j].dev)          &&          (!strcmp(nets-
>net[i].conex[j].function,"IN")))
                {
```

```
                //Buscamos posicion en la cadena JTAG
                for(z=0;z<pcb->numic;z++)
                {
                if(!strcmp(nets->net[i].conex[j].dev,pcb->chain[z].name))
                        {
                                id=z+48;
                                break;
                        }
                }
                if(!strcmp(value,"1"))
                        {
                fprintf(fp,"%c %s %s %s\t",id, nets->net[i].conex[j].pin,nets-
>net[i].conex[j].function,value);
                        flag=1;
                        }
                if(strcmp(value,"1") && strcmp(nets->net[i].conex[j].value,"X"))
                        {
                fprintf(fp,"%c %s %s %s\t",id, nets->net[i].conex[j].pin,nets-
>net[i].conex[j].function,nets->net[i].conex[j].value);
                        flag=1;
                        }
                }
        }
        fprintf(fp,"EVT");
        if(flag) cont++;
}
        fprintf(fp,"\nENDTest_Vector");
        por=(cont*100)/i;
        fprintf(fp,"\n\nNets Testeadas %d de %d (%d%c)",cont,i, por,'%');
}

int is_JTAG(pcb_t *pcb,char name[N])  // Devuelve 1 si es un dispositivo de la
cadena, 0 si no lo es
{
        int i;
        int flag=0;

        for(i=0;i<pcb->numic;i++)
        {
                if(!strcmp(pcb->chain[i].name,name))
                {
                        flag=1;
                }

        }
        return(flag);
}
```

2 JTEST-B

2.1 Fichero de cabecera (*jtest.h*)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define N 20
#define M 5
#define C 3000
#define L 100

typedef struct
{
	int id;  // 0,1,2,3...
	char name[N];		// ICXX
	char part_name[N]; // XC3S1500_FG676 .....
}ic_t;

typedef struct
{
	char pcb_version[M];		// Version PCB
	char pcb_name[N]; //Nombre del PCB
	int numic;			// Total de ic en la cadena JTAG
	ic_t chain[N];			// Una chain esta formada por varios ic

}pcb_t;

typedef struct
{
	int dev;		// 0,1,2,3...
	char port[N]; // Port Name IO_XX...
	int pin;// Celula de la cadena JTAG (posicion)
	int net;    // Net ID
	char function[M]; // IN, OUT
	char value[M]; // X, 0, 1
	char net_name[N];  // netname
}conex_t;

typedef struct
{
	conex_t conex[C];
	int num;			//Total conexiones

}initial_t;
```

```
typedef struct                  //Estructura resultado net test
{
        int net;        // Net ID 0,1,2,3.....
        int dev;        // 0,1,2,3...
        char port[N]; // Port Name IO_XX...
        int test_level;// Nivel insertado en un OUT_PORT "0","1"
        int read_level;         // Nivel leido en un INPUT_PORT  "0","1"
        char status[M];         // PASS o FAIL
        char net_name[N];  // netname

}test_t;

typedef struct                  // Array de Tests
{
  int num;
  int total;
  int test;
  test_t result[C];

}result_t;

void init_chain(pcb_t *pcb);
void load_file(pcb_t *pcb);
void ad_ic(pcb_t *pcb, ic_t *ic, int id, char name[N], char part_name[N]);
void print_chain(pcb_t *pcb);
int check_devices(pcb_t *pcb);
void ad_conex(pcb_t *pcb, initial_t *vect,int dev,char port[N],char
function[M],char value[M],int net,char net_name[N]);
void init_vect(initial_t *vect);
void print_vector(initial_t *vect);
void test(pcb_t *pcb, initial_t *out,initial_t *inp,initial_t *outaux, initial_t
*inpaux1,initial_t *inpaux2,int net ,result_t *result);
void print_result(result_t *result);
static int setBitValue (char *str);
static int showBitValue (char *str);
```

2.2 Código fuente de la rutina *JTEST-B*

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>

#include <shell.h>
#include <jtag.h>
#include <jtest.h>
#include <jtd.h>

extern int         curdev, numdev;
extern int         adapterSet;
extern char          adapter[];
extern jchain_t     jch;
extern jtagdev_t    devices[];

int doJtest(shell_t *shell, int argc, char *argv[])
{
      pcb_t pcb;
      ic_t ic;
      FILE *fp;
      FILE *fs;
      initial_t out;
      initial_t inp;
      initial_t outaux;
      initial_t inpaux1;
      initial_t inpaux2;
      result_t result;


         int dev;         // DEVICE
         char port[N];  // PIN
         char function[M]; // IN, OUT, UNDEF
         char value[M]; // X, 0, 1


         char dato[N];
         char name[N];
         char part_name[N];
         char net_name[N];
         int id;
         int i;
         int net=-1;
         int stat;
         int cont1=0;
         int cont2=0;
         char file[M];
         int flag;
```

```
//Inicializamos estructuras de datos

        init_chain(&pcb);
        init_vect(&out);
        init_vect(&inp);
        init_vect(&outaux);
        init_vect(&inpaux1);
        init_vect(&inpaux2);

        result.num=0;


// Carga BSDL FILES, scanea JTAG chain, lista los devices...

        doScan(shell, 0, NULL);

        /*printf("\nIntroduce el nombre del fichero de Test: ");
        scanf("%s",file); */

                fp= fopen("Nextgag.vt","r");  // Cambiar luego por "file"
                if(fp == NULL)
                {
                        printf("\nError abriendo %s", file);
                        exit(0);
                }

                fs= fopen("result.log","w");  // Fichero de resultados
                if(fs == NULL)
                {
                        printf("\nError abriendo %s", file);
                        exit(0);
                }

        // Guardamos identificador de placa

                fscanf(fp,"%s %s", name,dato);
                strcpy(pcb.pcb_name,name);
                strcpy(pcb.pcb_version,dato);
```

```
        // Guardamos la configuracion de la cadena JTAG

        while(strcmp(dato,"ENDDevice") && !feof(fp))
                {
                        fscanf(fp,"%s", dato);
                        if(!strncmp(dato,"IC",2) )
                        {
                                strcpy(name,dato);
                                fscanf(fp,"%s", dato);
                                id=atoi(dato);
                                fscanf(fp,"%s", dato);
                                strcpy(part_name,dato);
                                ad_ic(&pcb, &ic,id,name,part_name);
                        }
                }

        // Inicializamos fichero de resultados

                fprintf(fs,"%s\t%s",pcb.pcb_name,pcb.pcb_version);

                fprintf(fs,"\nDevice\tID\tSpecification");

         // Guardamos la configuracion de la cadena JTAG
                for(i=0;i<pcb.numic;i++)
                {
                fprintf(fs,"\n%s\t%d\t%s",pcb.chain[i].name,          pcb.chain[i].id,
pcb.chain[i].part_name);
                }
                fprintf(fs,"\nENDDevice\n");
                fprintf(fs,"\nNET\tSTATUS\tDEV\tPORT\tTEST\tREAD");
                fclose(fs);

        // Comprobamos que la Cadena JTAG es correcta, según datos del
fichero de entrada, y comenzamos TEST

        if(check_devices(&pcb) != JTAG_SUCCESS)
        {

    //Ponemos los IC en modo "EXTEST"

           for(i=0;i < numdev ; i++)
           {
                   curdev=i;
                   doExtest(shell, 0, NULL);
           }
        curdev=-1;
        doExecute(shell, 1, NULL);

           printf("\nTesting....");
```

```
        // Guardamos el vector inicial Outs

                while(strcmp(dato,"Initial_OUTs") && !feof(fp))
                        {
                        fscanf(fp,"%s", dato);
                        }

                while(!feof(fp))
                        {
                                fscanf(fp,"%s",dato);
                                if(!strcmp(dato,"ENDOuts")) break;
                                dev=atoi(dato);
                                fscanf(fp,"%s",port);
                                fscanf(fp,"%s",function);
                                fscanf(fp,"%s",value);
                                ad_conex(&pcb,&out,dev,  port,  function,
value,net,net_name);
                         }

        // Guardamos el vector inicial Inps

                while(strcmp(dato,"Initial_INPs") && !feof(fp))
                        {
                        fscanf(fp,"%s", dato);
                        }
                while(!feof(fp))
                        {
                                fscanf(fp,"%s",dato);
                                if(!strcmp(dato,"ENDInps")) break;
                                dev=atoi(dato);
                                fscanf(fp,"%s",port);
                                fscanf(fp,"%s",function);
                                fscanf(fp,"%s",value);
                                ad_conex(&pcb,&inp,dev,  port,  function,
value,net,net_name);
                        }

        // Realizamos una primera iteracion de test, para compensar el retardo
entre out-inp
                inpaux1=inp;
                test(&pcb,&out, &inp, &outaux, &inpaux1, &inpaux2,net,&result);
                printf("\n");
```

```
// Leemeos cada uno de los vectores de test y almacenamos en registros auxiliares

while(strcmp(dato,"Test_vector") && !feof(fp))
        {
                fscanf(fp,"%s", dato);
        }
while(!feof(fp))
        {
        fscanf(fp,"%s", dato);
        if(!strcmp(dato,"ENDTest_Vector")) break;

        fscanf(fp,"%s",dato);
        net=atoi(dato);
         fscanf(fp,"%s",net_name);
         cont1++;
        flag=0;
        fscanf(fp,"%s",dato);

        while(strcmp(dato,"EVT"))
        {
                dev=atoi(dato);
                fscanf(fp,"%s",port);
                fscanf(fp,"%s",function);
                fscanf(fp,"%s",value);
                if(!strcmp(function,"OUT"))
                {
                        ad_conex(&pcb,&outaux,dev, port, function, value,net, net_name);
                        flag=1;
                }
                else
                {
        ad_conex(&pcb,&inpaux1,dev, port, function, value,net,net_name);
                        flag=1;
                }
                fscanf(fp,"%s",dato);
        }

        //Ejecutamos vector de test
                if(flag!=0)
                   {
        test(&pcb,&out, &inp, &outaux, &inpaux1, &inpaux2,net,&result);
                     cont2++;
                      }
        }
```

```
        // Realizamos un test adicional para compensar el desfase input-ouput

        test(&pcb,&out, &inp, &outaux, &inpaux1, &inpaux2,net,&result);

    result.total=cont1;
    result.test=cont2;
    print_result(&result);
    fclose(fp);
    }
}

void init_chain(pcb_t *pcb)

{
        // Inicializa estructura PCB
        pcb->numic=0;
}
void ad_ic(pcb_t *pcb, ic_t *ic, int id, char name[N], char part_name[N])
{
        //Añadimos IC
        pcb->chain[pcb->numic].id=id;
        strcpy(pcb->chain[pcb->numic].name,name);
        strcpy(pcb->chain[pcb->numic].part_name,part_name);
        pcb->numic++;
}
void print_chain(pcb_t *pcb)
{
        int i;

   printf("\n");
        printf("   #  Description\n");
        printf("  --  ------------------\n");

        for(i=0;i<pcb->numic;i++)
        {
                printf("  %2d    %s\n",pcb->chain[i].id,pcb->chain[i].part_name);
        }
   printf("\n");
}

int check_devices(pcb_t *pcb)
{
       int i;
       int flag=1;
       char      name[JTAG_NAME_SIZE+1];

    if(numdev==pcb->numic)
       {
       for(i=0;i<pcb->numic;i++)
```

```
                {
                JTAG_Get_Name(&jch, i, name);
                if(strcmp(name,pcb->chain[i].part_name))
                flag=0;
                }
        }
  else flag=0;

 if(!flag)
        {
printf("\nCadena JTAG incorrecta. Se esperaba la siguiente configuracion.");
print_chain(pcb);
        }

return(flag);

}
void init_vect(initial_t *vect)
{
  vect->num=0;
}

void ad_conex(pcb_t *pcb,initial_t *vect,int dev,char port[N],char
function[M],char value[M], int net, char net_name[N])
{
   int pbit=-1;
   int j,stat;


   if(strcmp(value,"X"))  // si el valor es X, no tiene sentido añadir al test
        {

        vect->conex[vect->num].dev=dev;

//Añadimos IO_ para convertir PIN a PORT

                strcpy(vect->conex[vect->num].port,"IO_");
                strcat(vect->conex[vect->num].port,port);

// Para los puertos que no son del tipo IO_XXX (Mirar los BSDL files) ex. M0,
M1, DONE....

        //Para XC3S1500_FG676

        if(!strcmp(port,"C2"))
                {
                if(!strcmp(pcb->chain[dev].part_name,"XC3S1500_FG676"))
                strcpy(vect->conex[vect->num].port,"HSWAP_EN_C2");
                }
```

```
        else if (!strcmp(port,"AD26"))
                {
                if(!strcmp(pcb->chain[dev].part_name,"XC3S1500_FG676"))
                strcpy(vect->conex[vect->num].port,"CCLK_AD26");
                }

        else if (!strcmp(port,"AC24"))
                {
                if(!strcmp(pcb->chain[dev].part_name,"XC3S1500_FG676"))
                strcpy(vect->conex[vect->num].port,"DONE_AC24");
                }

        else if (!strcmp(port,"AE3"))
                {
                if(!strcmp(pcb->chain[dev].part_name,"XC3S1500_FG676"))
                strcpy(vect->conex[vect->num].port,"MO_AE3");
                }

        else if (!strcmp(port,"AC3"))
                {
                if(!strcmp(pcb->chain[dev].part_name,"XC3S1500_FG676"))
                strcpy(vect->conex[vect->num].port,"M1_AC3");
                }

        else if (!strcmp(port,"AF3"))
                {
                if(!strcmp(pcb->chain[dev].part_name,"XC3S1500_FG676"))
                strcpy(vect->conex[vect->num].port,"M2_AF3");
                }

// XC2C250_FG456

        else if (!strcmp(port,"B3"))
                {
                if(!strcmp(pcb->chain[dev].part_name,"XC2C250_FG256"))
                strcpy(vect->conex[vect->num].port,"HSWAP_EN_B3");
                }

        else if (!strcmp(port,"Y19"))
                {
                if(!strcmp(pcb->chain[dev].part_name,"XC2C250_FG256"))
                strcpy(vect->conex[vect->num].port,"CCLK_Y19");
                }

        else if (!strcmp(port,"AB20"))
                {
                if(!strcmp(pcb->chain[dev].part_name,"XC2C250_FG256"))
                strcpy(vect->conex[vect->num].port,"DONE_AB20");
                }
```

```
        else if (!strcmp(port,"AB2"))
                {
                if(!strcmp(pcb->chain[dev].part_name,"XC2C250_FG256"))
                strcpy(vect->conex[vect->num].port,"MO_AB2");
                }

        else if (!strcmp(port,"W3"))
                {
                if(!strcmp(pcb->chain[dev].part_name,"XC2C250_FG256"))
                strcpy(vect->conex[vect->num].port,"M1_W3");
                }

        else if (!strcmp(port,"AB3"))
                {
                if(!strcmp(pcb->chain[dev].part_name,"XC2C250_FG256"))
                strcpy(vect->conex[vect->num].port,"M2_AB3");
                }


        strcpy(port,vect->conex[vect->num].port);
        strcpy(vect->conex[vect->num].function,function);
        strcpy(vect->conex[vect->num].net_name,net_name);
        vect->conex[vect->num].net=net;
        strcpy(vect->conex[vect->num].value,value);

   (vect->num)++;
        }
}

void print_vector(initial_t *vect)
{
    int i;

   for(i=0;i<vect->num;i++)
   {
   printf("\n %d %s %s %s",vect->conex[i].dev,vect->conex[i].port,vect-
>conex[i].function,vect->conex[i].value);
   }
}
```

2.3 Código fuente de la subrutina *TEST*

```
void test(pcb_t *pcb, initial_t *out,initial_t *inp,initial_t *outaux, initial_t *inpaux1,initial_t *inpaux2,int net,result_t *result)
{

        int i,j;
        int pbit;
        int dev;
        char str[N];
        int stat;
        int level;
        int flag;



//Cargamos vector incial de ouputs

for(i=0;i<out->num;i++)
{
curdev=out->conex[i].dev;
//pbit=out->conex[i].pin;
strcpy(str,out->conex[i].port);
strcat(str,"=");
strcat(str,out->conex[i].value);

stat=setBitValue(str);
if (!stat)
printf("\nError cargando port %s a nivel %s", out->conex[i].port,out->conex[i].value);
}

 //Cargamos Vectores de Test
 for(i=0;i<outaux->num;i++)
{
        curdev=outaux->conex[i].dev;
        //pbit=out->conex[i].pin;
        strcpy(str,outaux->conex[i].port);
        strcat(str,"=");
        strcat(str,outaux->conex[i].value);

        stat=setBitValue(str);
        if (!stat)
        printf("\nError cargando port %s a nivel %s", outaux->conex[i].port,outaux->conex[i].value);
}
```

```
// Inicializa de nuevo outaux
outaux->num=0;

//Ejecutamos un shift

if ((stat = JTAG_Shift(&jch)) != JTAG_SUCCESS)
        {
        printf("Shift error : %s\n", JTAG_StrError(stat));
        }


//Comparamos el vector retornado con el vector de inputs

for(i=0;i<inp->num;i++)
        {
                flag=0;
                curdev=inp->conex[i].dev;
                //pbit=out->conex[i].pin;
                strcpy(str,inp->conex[i].port);


                level=showBitValue(str);

        // Si hay algun valor incorrecto, lo guardamos en results
         if(level!=(atoi(inp->conex[i].value)))
                {
                result->result[result->num].test_level=atoi(inp->conex[i].value);
                result->result[result->num].read_level=level;
                strcpy(result->result[result->num].status,"FAIL");
                result->result[result->num].net=net;
                result->result[result->num].dev=curdev;
                strcpy(result->result[result->num].port,inp->conex[i].port);
                strcpy(result->result[result->num].net_name,inp-
>conex[i].net_name);
                flag=1;
                result->num++;
                }
        }
```

```
// Todos los valores son correctos
if(flag==0)
                {
                strcpy(result->result[result->num].status,"PASS");
                result->result[result->num].net=net;

                flag=0;
                result->num++;
                }


//Cargamos en el vector de inputs el contenido de inpaux2

for(i=0;i<inpaux2->num;i++)
        {
        for(j=0;j<inp->num;j++)
                {
        if((inpaux2->conex[i].dev==inp->conex[j].dev)      &&      !strcmp(inpaux2-
>conex[i].port,inp->conex[j].port))
                        {
                strcpy(inp->conex[j].function,inpaux2->conex[i].function);
                strcpy(inp->conex[j].value,inpaux2->conex[i].value);
                 strcpy(inp->conex[j].net_name,inpaux2->conex[i].net_name);
                inp->conex[j].net=inpaux2->conex[i].net;
                break;
                        }
                }

        }

// Inicializamos de nuevo inputaux2
inpaux2->num=0;

//Cargamos en inpaux2 los valores de inputs que apuntan inpaux1

for(i=0;i<inpaux1->num;i++)
        {
        for(j=0;j<inp->num;j++)
                {
                if((inpaux1->conex[i].dev==inp->conex[j].dev) && !strcmp(inpaux1-
>conex[i].port,inp->conex[j].port))
                        {
                        inpaux2->conex[i].dev=inpaux1->conex[i].dev;
                        inpaux2->conex[i].net=inpaux1->conex[j].net;
                        strcpy(inpaux2->conex[i].port,inpaux1->conex[i].port);
                        strcpy(inpaux2->conex[i].function,inp->conex[j].function);
                        strcpy(inpaux2->conex[i].value,inp->conex[j].value);
                        strcpy(inpaux2->conex[i].net_name,inp-
>conex[j].net_name);
```

```
                                        inpaux2->num++;
                                        break;
                        }
                }
        }

//Cargamos inpaux1 en inputs
for(i=0;i<inpaux1->num;i++)
        {
        for(j=0;j<inp->num;j++)
                {
                if((inpaux1->conex[i].dev==inp->conex[j].dev) && !strcmp(inpaux1-
>conex[i].port,inp->conex[j].port))
                        {
                        strcpy(inp->conex[j].function,inpaux1->conex[i].function);
                        strcpy(inp->conex[j].value,inpaux1->conex[i].value);
                         strcpy(inp->conex[j].net_name,inpaux1-
>conex[i].net_name);

                        inp->conex[j].net=inpaux1->conex[i].net;
                        break;
                        }
                }

        }

// Inicializamos de nuevo inputaux2
inpaux1->num=0;
}

void print_result(result_t *result)
{
        int i,j;
        float x;
        FILE *fs;
        int flag=0;

        fs= fopen("result.log","a");
                                if(fs == NULL)

                        {
                                printf("\nError abriendo result");
                                exit(0);
                        }

        for(i=1;i<result->num;i++)
        {
                if(!strcmp(result->result[i].status,"PASS"));
```

```
//fprintf(fs,"\n%s\t%s\t%d\t%s\t%d\t%d",result->result[i].net_name,result-
>result[i].status,result->result[i].dev,result->result[i].port,result-
>result[i].test_level,result->result[i].read_level);

else
{
        if(flag==0) printf("\nNET\tSTATUS\tDEV\tPORT\tTEST\tREAD\n");

fprintf(fs,"\n%s\t%s\t%d\t%s\t%d\t%d",result->result[i].net_name,result-
>result[i].status,result->result[i].dev,result->result[i].port,result-
>result[i].test_level,result->result[i].read_level);

printf("%s\t%s\t%d\t%s\t%d\t%d\n",result->result[i].net_name,result-
>result[i].status,result->result[i].dev,result->result[i].port,result-
>result[i].test_level,result->result[i].read_level);
                        flag++;
        }
}
   x=(result->test)*100/(result->total);

   printf("\nTesteadas %d nets de %d (%.1f%)",result->test,result->total,x);
   fprintf(fs,"\n\nTesteadas %d nets de %d (%.1f%)",result->test,result->total,x);

        if(flag==0)
        {
        printf("\n\nTest OK\n");
        fprintf(fs,"\n\nTest OK\n");
        }
        else
        {
                printf("\n\nTest FAIL\n");
                fprintf(fs,"\n\nTest FAIL\n");
        }

        fclose(fs);

}
```

```
static int doExec (void)
{
	char	cmd[256];
	int		i, stat;


	/* build the command-line */
	*cmd = '\0';
	for (i = 0; i < numdev; i++)
	{
		if (i > 0)
		strncat(cmd, ",", 256);
		strncat(cmd, devices[i].inst, 256);
	}

	return (JTAG_Execute(&jch, cmd));
}

static int setBitValue (char *str)
{
	int		j, stat, pbit, level;
	char	*val;
	char	pname[JTAG_NAME_SIZE+1];


	/* value is after the = sign */
	if ((val = strchr(str, '=')) == NULL)
	{
		printf("Expression must be : port = value ...\n");
		return 0;
	}

	/* get the pin/port name */
	strncpy(pname, str, val - str);
	pname[val-str] = '\0';

	/* get the value */
	while (*val == ' ' || *val == '=')
		val++;
```

```
/* query a pin name, then a port name, preferrably output */
for (j = 0; j < 5; j++)
{
switch (j)
        {
        case 0 :
        stat = JTAG_Query_Pin(&jch, curdev, pname, PORT_OUTPUT, &pbit);
        break;

        case 1 :
        stat = JTAG_Query_Port(&jch, curdev, pname, PORT_OUTPUT, &pbit);
        break;

        case 2 :
        pbit = -1;
        stat = JTAG_Query_Pin(&jch, curdev, pname, PORT_UNKNOWN,
&pbit);
        break;

        case 3 :
        pbit = -1;
        stat = JTAG_Query_Port(&jch, curdev, pname, PORT_UNKNOWN,
&pbit);
        break;

        case 4 :
                if (*str == '.')
                {
                        /* a direct bit index */
                        pbit = atoi(str+1);
                        stat = JTAG_SUCCESS;
                }
                else
                        stat = JTAG_ERR_PIN;
                        break;
                }

        if (stat == JTAG_SUCCESS)
                break;
        }

        if (stat != JTAG_SUCCESS)
        {
                printf("Query error for %s : %s\n", str, JTAG_StrError(stat));
                return 0;
        }
```

```
	/* set the pin value */
	if ((stat = JTAG_Set_Level(&jch, curdev, pbit, *val)) != JTAG_SUCCESS)
	{
		printf("Set error : %s\n", JTAG_StrError(stat));
		return 0;
	}

	return 1;
}

static int showBitValue (char *str)
{
	int		j, stat, pbit, level;


/* query a pin name, then a port name, preferrably input */
for (j = 0; j < 5; j++)
{
switch (j)
{
	case 0 :
	stat = JTAG_Query_Pin(&jch, curdev, str, PORT_INPUT, &pbit);
	break;

	case 1 :
	stat = JTAG_Query_Port(&jch, curdev, str, PORT_INPUT, &pbit);
	break;

	case 2 :
	pbit = -1;
	stat = JTAG_Query_Pin(&jch, curdev, str, PORT_UNKNOWN, &pbit);
	break;

	case 3 :
	pbit = -1;
	stat = JTAG_Query_Port(&jch, curdev, str, PORT_UNKNOWN, &pbit);
	break;

	case 4 :
		if (*str == '.')
		{
			/* a direct bit index */
			pbit = atoi(str+1);
			stat = JTAG_SUCCESS;
		}
		else
			stat = JTAG_ERR_PIN;
			break;
		}
```

```
                if (stat == JTAG_SUCCESS)
                        break;
        }

        if (stat != JTAG_SUCCESS)
        {
                printf("Query error for %s : %s\n", str, JTAG_StrError(stat));

        }

        /* get the pin value */
        if ((stat = JTAG_Get_Level(&jch, curdev, pbit, &level)) !=
JTAG_SUCCESS)
        {
                printf("Error : %s\n", JTAG_StrError(stat));
        }

        return(level);
}
```